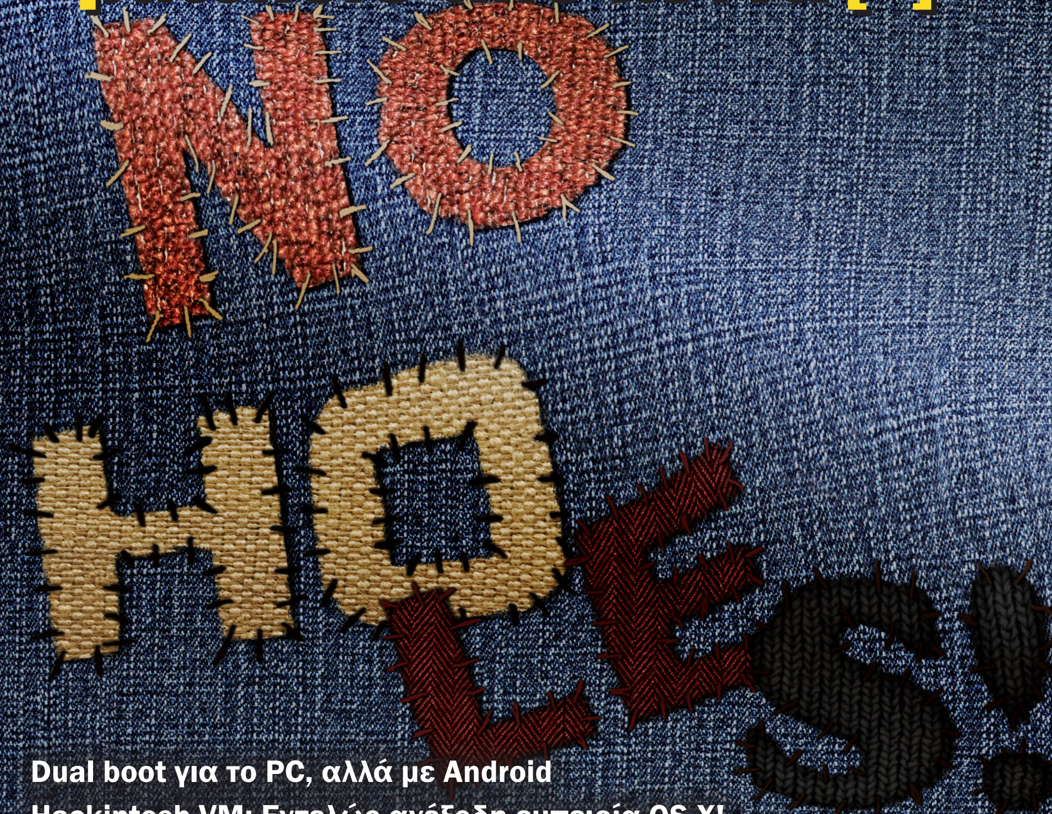
all rights reversed

HACKER

Ιούνιος 2014 # Τιμή τεύχους 4€
τεύχος 033 - Crypto-Summer Edition

Οι θαυματουργοί patchers από κοντά [*]



Dual boot για το PC, αλλά με Android

Hackintosh VM: Εντελώς ανέξοδη εμπειρία OS X!

Γεννήτριες πολυμορφικού κώδικα κρυπτογράφησης

Φτιάξτε πραγματικά RNGs, με το παλιό ραδιοφωνάκι

[*] ...και καλή τύχη στους επίδοξους crackers!

HACKER

Περιεχόμενα

και οδηγίες χρήσης

Εκδότης / Διευθυντής Χρήστος Βαρελάς

Αρχισυντάκτης Παναγιώτης Βαρελάς

Σύμβουλοι έκδοσης Νίκος Μουρατίδης, Βούλα Παυλίδου

Σύνταξη Γιώργος Αραβίδης, Παναγιώτης Βαρελάς, Χρήστος Βαρελάς, Ανδρέας Βενιέρης, Γιώργος Γιάννου, Χρήστος Θεμελής, Μανώλης Κιαγιάς, Ιωάννης Κονιάρης, Πέτρος Κυλαδίτης, Γιώργος Μελέκος, Νίκος Μουρατίδης, Νικόλαος Σηφάκης, Μέσος Παπαδόπουλος (TM), Χρίστος Τόμπρας

Γραμματική επιμέλεια Βούλα Παυλίδου

Καλλιτεχνική Επιμέλεια Πέτρος Φιλιππίδης

Το ηλεκτρονικό περιοδικό deltaHacker είναι συνδρομητικό και εκδίδεται από την Parabing Creations δώδεκα φορές το χρόνο, κάθε μήνα. Η Parabing Creations έχει την έδρα της στη Μαιώτιδος 3, 55133, Καλαμαριά. Ο δικτυακός τόπος του περιοδικού είναι στο http://deltahacker.gr. Το email επικοινωνίας είναι το talk2us@deltahacker.gr και ειδικά για τις συνδρομές είναι το subscriptions@deltahacker.gr. Οι γενικές πληροφορίες για τις συνδρομές παρατίθενται στο http://deltahacker.gr/subscriptions και οι ίδιες οι παραγγελίες γίνονται από το http://deltahacker.gr/pdf12order. Όλες οι απόψεις που εκφράζονται στα άρθρα δεν εκφράζουν απαραίτητα και τη γνώμη του περιοδικού.

Πιθανότητα: Είστε συνένοχοι στη διάπραξη παρανομιών

Αγαπητές κι αγαπητοί,

- Τώρα που σας γράφω, σύμφωνα με σχετικά πρόσφατη συνήθειά μου κανονικά θα έπρεπε να σας γράψω τι συμβαίνει τώρα, που σας γράφω.
- Πρόκειται για ένα απλό κι αποτελεσματικό τρικ ώστε να ξεκινήσεις ομαλά μια στήλη σαν αυτή, πριν ακόμα περάσεις στα κύρια ζητήματα που σε απασχολούν ή σε αυτά που θέλεις να επικοινωνήσεις (sic), τέλος πάντων.
- Αλλά όταν τα χρησιμοποιείς συχνά-πυκνά, ακόμη και τα πιο αποτελεσματικά τρικ ξεφτίζουν και χάνουν την αξία τους.
- Ακριβώς γι' αυτό, αγαπητές κι αγαπητοί, αποφάσισα να μη σας γράψω τι ακριβώς συμβαίνει τώρα, που σας γράφω.
- Μπορώ βέβαια να σας γράψω τι συνέβη το πρωί, όταν ακόμα ήμουν στον πρώτο καφέ και δεν έγραφα και λέγαμε με την κα Σύμβουλο Εκδόσεως™, που κι εκείνη ήταν στον πρώτο καφέ, ότι θέλουμε να καλέσουμε τον κο Αρχισυντάκτη™ προς τα μέρη μας, και μετά βγήκε βιαστικά για μια βόλτα —η κα Σύμβουλος Εκδόσεως™, όχι ο κος Αρχισυντάκτης™— "πριν δυναμώσει η ζέστη", όπως σημείωσε, αλλά εγώ έμεινα να χαζεύω στο ίντερνετ γιατί, πώς να το κάνουμε, για 'μένα η ζέστη είναι εξ ορισμού δυναμωμένη το καλοκαίρι και υπό αυτές τις συνθήκες δεν μου είναι καθόλου εύκολο να πέφτω με τα μούτρα στη δουλειά.
- Χάζευα στο ίντερνετ, που λέτε, και δεν είχα ακόμα αποφασίσει για το κεντρικό θέμα αυτού του εντιτόριαλ. Εντάξει, μερικά υποψήφια θέματα τα είχα κατά νου.
- Να, σκεφτόμουν για παράδειγμα να εξηγήσω γιατί το περιοδικό deltaHacker δεν είναι σε ανταγωνισμό με κανένα από τα ντόπια τεχνολογικά περιοδικά και sites.
- Για 'μένα και τους συνεργάτες του deltaHacker είναι φανερό γιατί δεν συμβαίνει αυτό. Για άλλους όμως, δεν είναι. Ούτως ή άλλως λειτουργούν εντελώς διαφορετικά οι άνθρωποι, επομένως εντελώς διαφορετική είναι και η οπτική τους κι ο τρόπος σκέψης τους κι τρόπος δράσης τους και τα πάντα όλα τους. Άντε τώρα, λοιπόν, να μπαίνεις σε μια διαδικασία κατά την οποία προσπαθείς, κατ' ουσίαν, ν' αποδείξεις ότι δεν είσαι Ελέφαντας.
- Να με συγχωρείτε πολύ, αλλά καλοκαιριάτικα βαριέμαι ν' αποδεικνύω ότι δεν είμαι Ελέφαντας.
- Τώρα που το σκέφτομαι καλύτερα, το ίδιο βαριέμαι να το αποδεικνύω και τις άλλες εποχές του χρόνου.
- Τέλος πάντων.

- Ένα άλλο θέμα που είχα κατά νου για εντιτόριαλ, αφορούσε σ' αυτή τη διακαή επιθυμία που έχει πολύς, κατά πώς φαίνεται, κόσμος, για να "σπάει κωδικούς φέισμπουκ". Μπορεί να κάνω λάθος εδώ, πλέον όμως τείνω να πιστεύω ότι κανείς νεαρός δεν εμπιστεύεται καμία νεαρά — και το αντίστροφο. Τι να πω, περνάνε κρίση οι ανθρώπινες σχέσεις κι αυτό το θυμάμαι από τότε που θυμάμαι τον εαυτό μου — άντε 15 χρόνια αργότερα, δηλαδή.
- Εκεί λοιπόν που χάζευα στο ίντερνετ κι απέρριπτα θέματα για εντιτόριαλ και η ζέστη δυνάμωνε αλλά όχι για 'μένα αφού είναι ήδη δυναμωμένη, έπεσα πάνω σε μια είδηση που είχε διπλό αντίκτυπο στον μικρόκοσμό μου, τουλάχιστον για εκείνη ακριβώς τη στιγμή (ξέρετε, που χάζευα στο ίντερνετ κι απέρριπτα θέματα για εντιτόριαλ και η ζέστη δυνάμωνε — αλλά όχι για 'μένα): α) θυμήθηκα τα πρότζεκτ με τα οποία καταπιάνομαι αυτόν τον καιρό, β) βρήκα αμέσως κεντρικό θέμα για το εντιτόριαλ που *αυτή* τη στιγμή ακόμα γράφω αλλά δεν το έχω ολοκληρώσει και για την ακρίβεια έχω φτάσει ως αυτήν εδώ την τελεία → .
- Πριν από τρία περίπου χρόνια, που λέτε, είχαν κατασχεθεί οι υπολογιστές ενός αυστριακού διαχειριστή Tor exit node. Κάποιος έκανε κάτι παράνομο στο ίντερνετ (σιγά το νέο) και για να κρύβει τα ίχνη του χρησιμοποιούσε το δίκτυο ανωνυμίας του Tor (επίσης σιγά το νέο). Οι Αρχές κινητοποιήθηκαν και μετά από σχετική έρευνα δεν έφτασαν σ' εκείνον που διέπραξε την παρανομία, αλλά στον αυστριακό ιδιοκτήτη του υπολογιστή σε ρόλο Tor exit node. Μέχρι εκεί μπόρεσαν να "δουν" οι Αρχές, άρα θεώρησαν λογικό ότι το υλικό για τη "διάπραξη της παρανομίας" πρέπει να κατασχεθεί κι ο ιδιοκτήτης του να δώσει εξηγήσεις στη δικαιοσύνη.
- Μαντέψτε τι έγινε πρόσφατα.
- Αυστριακό δικαστήριο έκρινε το φίλο μας ένοχο, ως συνεργό εκείνου που διέπραξε την παρανομία (και ποτέ δεν συνελήφθη). Η λογική του δικαστή ήταν ότι, σύμφωνα με την παράγραφο 12 του αυστριακού ποινικού κώδικα, η λειτουργία και μόνο του Tor exit node αποτελεί συνεισφορά στη διάπραξη παράνομης πράξης.
- Παράλογη απόφαση;
- Κάποιος μπορεί να πει ότι δεν είναι και τόσο παράλογη αφού, πράγματι, όσοι χρησιμοποιούν το δίκτυο του Tor κάτι έχουν να κρύψουν κι όσοι έχουν κάτι να κρύψουν συνήθως έχουν και κάποιον πονηρό σκοπό ή σκοτεινή πρόθεση. Από την άλλη, αυτοί που δεν έχουν τίποτε να κρύψουν δεν έχουν και κανένα λόγο για να χρησιμοποιούν το Tor. Επίσης: Εκείνοι που στήνουν και συντηρούν Tor nodes —σαν τον γράφοντα, καλή ώρα, που έχει δύο— γνωρίζουν πολύ καλά όλα τα προηγούμενα και μολαταύτα συνεχίζουν να λειτουργούν τους κόμβους κι άρα συνεχίζουν να υποβοηθούν τους διάφορους σκοτεινούς τύπους και τους ξεκάθαρα κακούρ-

γους, οπότε όλοι μαζί συμβάλλουν ώστε ο κόσμος μας να μην είναι τόσο φωτεινός κι ωραίος όσο θα μπορούσε να είναι, άρα δίκιο είχε ο αυστριακός δικαστής — εύγε και μπράβο και πάντα τέτοια, βρέ!

- Μη νομίζετε ότι τα βγάζω από το μυαλό μου, όλα αυτά. Τέτοια επιχειρήματα "παίζουν" συχνά και σε κάθε σχετική συζήτηση υπάρχει πάντοτε *τουλάχιστον* ένας που αμολάει το "για να θέλεις να κρύβεσαι κάτι δεν πάει καλά" επιχείρημα.
- Θα είχε πολύ ενδιαφέρον, πάντως, αν μπορούσαμε να βλέπαμε τους ίδιους αυτούς τύπους να προσπαθούν να εκφραστούν τη στιγμή που βρίσκονται πίσω από ένα "πολιτικό firewall", το οποίο όποιος επιχειρεί να διαπεράσει έχει να αντιμετωπίσει καθόλου ευχάριστες συνέπειες.
- Υπάρχουν βέβαια κι εκείνοι που χρησιμοποιούν την τεχνολογία για τη διάπραξη παρανομιών. Διορθώνω: Υπήρχαν, υπάρχουν και θα συνεχίσουν να υπάρχουν. Αλλά είναι ένα πράγμα η ύπαρξη μιας τεχνολογίας ή ενός εργαλείου, κι εντελώς άλλο η χρήση που ο καθένας κάνει.
- Εκείνοι δε που φροντίζουν για την ύπαρξη ή τη διάθεση των εργαλείων, τι ρόλο έχουν; Είναι δυνατόν να θεωρούνται συνυπεύθυνοι στη διάπραξη παρανομιών, όταν οι βασικοί υπαίτιοι έχουν κάνει χρήση των υπό συζήτηση εργαλείων;
- Ναι, αμέ, φυσικά και είναι δυνατόν να θεωρούνται συνυπεύθυνοι.
- Μόνο που τότε πρέπει να θεωρηθούν συνυπεύθυνοι κι όσοι φτιάχνουν και διαθέτουν αυτοκίνητα (τα χρησιμοποιούν οι ληστές), λάστιχα (φτιάχνουν σφεντόνες τα παιδιά και σπάνε τζάμια), πτυσσόμενες σκάλες (οι ληστές που λέγαμε) κι ένα σωρό άλλοι.
- Θέλω τώρα να κάνω μια μικρή εξομολόγηση: Από τότε που ο Snowden άρχισε τις αποκαλύψεις του, εγώ τουλάχιστον συνεχίζω να μην πολυαγχώνομαι. Μάλλον θα έχει να κάνει με την ιδιοσυγκρασία μου, όμως ακόμη και τώρα σκέφτομαι ότι δεν έχω μεγάλα μυστικά που πάση θυσία πρέπει να προστατεύω. Ίσως φταίει και η (όποια) μαθηματική παιδεία έχω πάρει, κατά τη διάρκεια των φοιτητικών ετών: Σκέφτομαι δηλαδή ότι η πιθανότητα να με παρακολουθούν ή/και να πέσω θύμα κάποιας πλεκτάνης είναι, πρακτικά, μηδέν. Δεν ζούμε δα στην Κίνα, ούτε στο Ιράν, ούτε στη Βόρειο Κορέα. Μη με παρεξηγείτε: Ξέρω ότι αδικίες συμβαίνουν κι εδώ, στον λεγόμενο πολιτισμένο κόσμο, όμως αρχίζω πάλι να υπολογίζω αυτές τις πιθανότητες και χαλαρώνω.
- Την ίδια στιγμή, θεωρώ τις δυνατότητες επιλογής για anonymity ή/και για privacy ως βασικά ανθρώπινα δικαιώματα.
- Δεν υπερβάλλω καθόλου, αγαπητές κι αγαπητοί.
- Η σκέψη και μόνο ότι κάποια στιγμή *δεν* θα έχω το δικαίωμα να κρύβω την ταυτότητά μου ή/και να κρύβω πράγματα που με αφορούν, κυριολεκτικά με τρελαίνει. Θεωρώ δε ότι μια τέτοια κοινωνία δεν θα είναι φτιαγμένη για ανθρώπους με ελεύθερη βούληση, αλλά για προγραμματιζόμενα ρομπότ-σκλάβους.

- Κάθομαι, λοιπόν, τους τελευταίους μήνες και παίζω. Να, έχω ας πούμε φτιάξει ένα δικό μου σύστημα cloud storage, ώστε να μην εξαρτώμαι από το Dropbox (που κατά τα άλλα λατρεύω). Αν έχετε τη διάθεση, δείτε πώς πήγε το πρότζεκτ —καθώς κι όλες τις λεπτομέρειες υλοποίησής του— σ' αυτά τα δύο video tutorials:

  http://deltahacker.gr/deltacast-s02e03
  και http://deltahacker.gr/deltacast-s02e04.
- Αν πάλι δεν σας ενδιαφέρει το cloud storage αλλά θα θέλατε να συνεισφέρετε κι εσείς στο δίκτυο ανωνυμίας του Tor, τότε ίσως πρέπει να δείτε την περίπτωση δημιουργίας ενός relay — αν όχι exit node: http://deltahacker.gr/?p=9583. Και ναι, είναι καλό να βοηθάμε την (παγκόσμια) κοινότητα, αλλά δεν πρέπει να ξεχνάμε να προστατεύουμε κι εαυτούς. Σας πιάνει άραγε κάποια ανησυχία, κάθε φορά που συνδέεστε σε δημόσιο HotSpot; Λογικό είναι! Υπάρχουν φορές που δεν εμπιστεύεστε ούτε τον ISP σας; Κατανοητό! Η λύση σ' αυτές τις περιπτώσεις (κι όχι μόνο) ακούει στο όνομα OpenVPN. Δείτε: http://deltahacker.gr/?p=5058
- Δεν μπορείτε να πείτε, ε; Για κάποιον που δεν πολυαγχώνεται για το τι μπορεί να του συμβεί στο ίντερνετ, το ψάχνω αρκετά.
- Κι έχει και συνέχεια.
- Για το επόμενο επεισόδιο του deltaCast (http://deltahacker.gr/category/deltacast) είχα σκοπό να βελτιώσω εκείνο το cloud storage system, βάζοντας στο μείγμα το encrypted LVM (είναι και που σταμάτησε η ανάπτυξη του TrueCrypt, βλέπετε). Την ίδια στιγμή όμως παρατηρώ ότι εδώ και καιρό έχει αναθερμανθεί το ενδιαφέρον για την κρυπτογράφηση του email. Προσέξτε: Δεν εννοώ την κρυπτογράφησή του από τον υπολογιστή μου ως τον mail server (π.χ., της Google ή της Yahoo). Εννοώ να είναι κρυπτογραφημένο ακόμη κι όταν βρίσκεται αποθηκευμένο στον mail server και κανείς να μην μπορεί να το διαβάσει *εκτός* από τον παραλήπτη. Έριξα λοιπόν την ιδέα για σχετικό video tutorial στη fan page του περιοδικού (https://facebook.com/deltaHacker) και είδα ότι, πράγματα, το encrypted LVM πρέπει να περιμένει λίγο και το επόμενο (ή τα δύο επόμενα) video tutorial πρέπει να αφορά στην κρυπτογράφηση του email.
- Και μετά από αυτό ξέρετε τι θα ακολουθήσει, έτσι δεν είναι; Ακριβώς: Η δημιουργία του δικού μας mail server, ώστε τα email μας να μην αποθηκεύονται καν στους servers τρίτων — ακόμη κι αν έχουμε φροντίσει ώστε να είναι ισχυρά κρυπτογραφημένα!
- Ξέρετε κάτι; Λένε ότι το καλοκαίρι είναι η εποχή για να χαλαρώνουμε και για να κάνουμε πράγματα που δεν είχαμε την ευκαιρία να κάνουμε όλη τη χρονιά.
- Αυτό ακριβώς λέω κι εγώ ;)

Καλά να περνάτε,
Ο Προκομμένος Σας ™

POST CARD

THE ADDRESS TO BE WRITTEN ON THIS SIDE





ουέλ, μέλ

Αγαπητοί μου,

Από το πρωί, σήμερα Κυριακή, είμαι καθισμένος στο παράθυρο και παρακολουθώ έναν κίτρινο γάτο απέναντι, και τις κινήσεις του.

Είμαι μόνος στο σπίτι. Ο ανιψιός μου έλεγε να πάμε μαζί για ψώνια αλλά εγώ βαριόμουν να βγω. Μισοβρέχει άλλωστε και δεν έχω όρεξη.

Παρακολουθώ λοιπόν τον γάτο. Δεν είναι απολύτως κίτρινος. Έχει και λίγο άσπρο, αλλά μου αρέσει να τον λέω κίτρινο.

Στην αρχή, όταν τον πρωτοείδα, ήταν καθισμένος σε ένα πεζούλι, δίπλα στην φυλλωσιά. Έμοιαζε ατάραχος, σίγουρος για τον εαυτό του, και δεν έδινε καμία σημασία στα αυτοκίνητα του δρόμου ή τους περαστικούς που τον προσπερνούσαν.

Κάποιοι σταματούσαν και του μιλούσαν, οπότε αυτός ανταπέδιδε τις φιλοφρονήσεις με χαρά.

Πιο μετά τον είδα να διασχίζει τον δρόμο για να περάσει απέναντι, ατάραχος πάντα, σίγουρος με τον εαυτό του, χαλαρός.

Τον διάσχισε διαγωνίως, taking his time, όπως λένε εδώ οι ντόπιοι, σε σημείο που ένα αυτοκίνητο που έτυχε να κατηφορίζει την ίδια στιγμή δεν είχε άλλη επιλογή από το να σταματήσει και να τον περιμένει.

Μου έκανε μεγάλη εντύπωση αυτό.

Μου θύμισε μια άλλη σκηνή, προ ημερών.

Ήμουν στο λεωφορείο 46 που κάνει στάση εδώ πιο δίπλα, και το οποίο παίρνω όποτε έχω να περάσω από την βιβλιοθήκη. Πηγαίναμε κούτσα κούτσα από κάτι στενά δρομάκια, και κάποια στιγμή συνειδητοποιώ ότι το λεωφορείο πήγαινε πολύ αργά, χωρίς όμως να έχει κίνηση. Κοίταξα παραξενεμένος από το παράθυρο και βλέπω μπροστά μας έναν τύπο με ποδήλατο που πάσχιζε να τρέξει όσο πιο γρήγορα μπορούσε. Αυτός πάσχιζε, μεν, αλλά και πάλι μας καθυστερούσε.

Αυτό που μου είχε κάνει εντύπωση ήταν η στωικότητα (πώς αλλιώς να την πω;) του οδηγού του λεωφορείου. Είχε κόψει ταχύτητα και ακολουθούσε τον ποδηλάτη περιμένοντας να ανοίξει λίγο ο δρόμος για να μπορέσει να τον προσπεράσει με ασφάλεια. Χωρίς βιασύνες, χωρίς ανυπομονησίες, χωρίς νεύρα. Και κυρίως χωρίς να χτυπάει το κλάξον και χωρίς να μαρσάρει την μηχανή του λεωφορείου για να τον ψαρώσει.

Απόρησα.

Θα μου πείτε, δεν είναι θέμα στωικότητας του συγκεκριμένου οδηγού. Αυτές θα είναι οι οδηγίες που έχει. Τις οδηγίες ακολουθεί.

Μα ναι, δεν αντιλέγω καθόλου. Τις οδηγίες ακολουθεί. Αλλά αυτό ακριβώς είναι που με κάνει και απορώ. Το ότι καταφέρνει να ακολουθεί τις οδηγίες χωρίς να το παίρνει προσωπικά. Πώς τα καταφέρνει;

Και, βεβαίως, ακόμα μεγαλύτερη εντύπωση μου κάνει το ότι παρόλο που οι οδηγίες είναι τέτοιες, το λεωφορείο φτάνει πάντα —ή σχεδόν πάντα—

στην ώρα του.

Και εν πάση περιπτώσει, σε κάθε στάση υπάρχει φωτεινή ένδειξη που σου λέει σε πόση ώρα πρόκειται να καταφτάσει ποιο λεωφορείο. Γενικώς, ποδηλάτης ξε-ποδηλάτης στον δρόμο, τα λεωφορεία προχωρούν, οι φωτεινές επιγραφές ενημερώνουν, ο κόσμος πάει στην δουλειά του, τα πράγματα δείχνουν να δουλεύουν, λίγο πολύ, ρολόι.

Αυτό λοιπόν με κάνει και απορώ. Πώς το έχουν πετύχει; Τι πίνουν;

Ξέρω, ξέρω, θα μου πείτε για την ιδιοσυγκρασία των Άγγλων, για την έμφυτη πειθαρχία τους, για το προτεσταντικό (ή, έστω, το αγγλικανικό) τους πνεύμα και λοιπά και λοιπά και λοιπά. Θα το αντιπαραθέσετε αυτό με το μεσογειακό μας ταπεραμέντο, την έμφυτη υπερηφάνεια μας και λοιπά και λοιπά και λοιπά.

Εύλογη η εξήγηση. Ισχύει;

Δεν ξέρω, ρε παιδιά, να το σκεφτούμε. Εμφυτη πειθαρχία, έστω. Αλλά έμφυτη σε ποιόν; Οι μισοί οδηγοί λεωφορείων στο Λονδίνο είναι Ινδοί και οι άλλοι μισοί Τζαμαϊκανοί. (Δεν αστειεύομαι. Υπάρχουν και μερικοί Εγγλέζοι, εντάξει. Αλλά είναι μειονότητα.) Πού την βρήκαν την έμφυτη πειθαρχία στην Ινδία ή στην Καραϊβική; Πείτε μου.

Εντάξει, στατιστική της κακιάς ώρας είναι αυτή, αλλά το ερώτημα παραμένει, μην μπερδευόμαστε.

Και δεν είναι μόνο αυτό.

Προ ημερών ήμουν με τα πόδια, και έψαχνα να βρω πώς θα πάω κάπου, αλλά είχα μισοχαθεί. Σταματάω δύο αστυνομικούς που έκαναν περιπολία να ρωτήσω. (Θα θυμάστε που σας έχω πει πως συμπαθώ τους Βρετανούς αστυνομικούς). Τους σταματάω, λοιπόν, και μου πιάνουν την κουβέντα. Ήταν μια γυναίκα και ένας άντρας. Καλωδιωμένοι, με τα κινητά τους, τους ασυρμάτους τους, χειροπέδες να κρέμονται από την ζώνη, άοπλοι κατά τα άλλα, και κυρίως χαμογελαστοί και εξυπηρετικότατοι. Η μία, η αστυνομικίνα, δεν ήταν Αγγλίδα. Το άκουγες στην προφορά της και μιας και μιλάγαμε τόσο φιλικά την ρώτησα. Ήταν Πολωνέζα, μου λέει, από την Κρακοβία. Ο άλλος



έμοιαζε Αφρικανός. Τον ρώτησα κι αυτόν. Αυτός ήταν Βρετανός, αλλά με καταγωγή από την Νιγηρία. Με ρωτήσανε βέβαια και εκείνοι, κι όταν τους είπα πως είμαι από την Ελλάδα καταχαρήκανε. Γελάγανε με την Αγγλία που αποκλείστηκε από το Παγκόσμιο Κύπελλο και μου ευχήθηκαν ό,τι καλύτερο για την Ελλάδα, που μόλις είχε περάσει στους 16. Είπαμε μια δυο βλακείες ακόμα και μετά συνεχίσαμε, οι μεν και οι δε, τους διαφορετικούς μας δρόμους.

Θα νομίζετε πως υπερβάλλω, αγαπητοί μου, κι ότι τα βλέπω όλα όμορφα. Δεν ξέρω, μπορεί να έχετε δίκιο. Μπορεί να μας έχουν βάλει κάτι στο νερό ή να μας ψεκάζουν.

Το ζήτημα ωστόσο είναι πως ό,τι κι αν έχουν βάλει στο νερό, μια τέτοια συζήτηση, ή έστω κάποια παρόμοια, δεν θα μπορούσα να με φανταστώ να την κάνω στην Αθήνα. Μικρή η πιθανότητα.

Το ερώτημα, λοιπόν, που με απασχολεί από το πρωί που παρακολουθώ τον κίτρινο γάτο να σουλατσάρει είναι αυτό: Ποιοι παράγοντες έχουν συντελέσει ώστε μια κοινωνία όπως η βρετανική να σκέφτεται σήμερα και να συμπεριφέρεται με τόσο οργανωμένο, ανοιχτό και διαλλακτικό τρόπο;

Ηταν η οικονομική της δύναμη; Η αποικιοκρατία; Ο ιμπεριαλισμός; Η παιδεία;

Τι το διαφορετικό συμβαίνει στην Γηραιά Αλβιώνα, και τι θα μπορούσαμε να διδαχτούμε από αυτούς;

Το ερώτημα δεν μου φαίνεται καθόλου απλό. Δεν είμαι καν σίγουρος ότι μπορώ να το αποτυπώσω στο χαρτί με σαφήνεια.

Αλλά να. Κάνω τις συγκρίσεις μου.

Ανοίγω ας πούμε αυτές τις μέρες το φέισμπουκ του ανηψιού και διατρέχω τις ελληνόφωνες διαδικτυακές συζητήσεις στις οποίες συμμετέχει.

Με πιάνει ζαλάδα. Όλοι την πέφτουν σε όλους. Μερικοί σου την πέφτουν για την άγνοιά σου. Σε κατακεραυνώνουν για την αδυναμία σου να κατανοήσεις την βαθύτερη ουσία του τάδε ή του δείνα προβλήματος, σε ειρωνεύονται, σε περιγελούν, βάζουν διαχωριστικές γραμμές, φωνάζουν, προειδοποιούν.

Άλλοι εισβάλλουν στη συζήτηση σαν ταύροι σε υαλοπωλείο. Υπογραμμίζουν συμπτώσεις, στηλιτεύουν ανθρώπους, αποκαλύπτουν συνωμοσίες, καταγγέλλουν σχέδια αφελληνισμού. Βάζουν και αυτοί διαχωριστικές γραμμές, φωνάζουν, προειδοποιούν.

Άλλοι πάλι έχουν ξεφύγει εντελώς. Επιτίθενται λεκτικά, προσβάλλουν, βρίζουν, ουρλιάζουν, απειλούν. Δεν αφήνουν καμία αμφιβολία πως αν σε βρίσκανε μπροστά σου θα σε έφερναν σε δύσκολη θέση.

Το "αστείο" βέβαια είναι ότι διαφορετικές συζητητικές αυτές μεθοδολογίες δεν έχουν σαφές correlation με τις ιδέες που διατυπώνονται. Όχι σπάνια συνειδητοποιώ, με θλίψη μου, πως ο έξαλλος γορίλας που βλέπω να χτυπιέται και να απειλεί απέναντί μου, υποστηρίζει πράγματα με τα οποία, υπό άλλες, πιο πολιτισμένες, συνθήκες θα μπορούσα και να συμφωνώ.

Και τότε νοιώθω να χάνω κάθε ελπίδα.

Αναρωτιέμαι τι πρέπει να κάνω.

Προς το παρόν παρακολουθώ τον κίτρινο γάτο.

Το βράδυ στις 9 θα δούμε το ματς με την Κόστα Ρίκα.

Για πιο μετά βλέπουμε.

Σας ασπάζομαι,
Θείος Ακάκιος

Skill: Advanced
Tags: relative virtual address, relocation table, import table, indirect calls

NO HOLES!

by Jef

Στη μνήμη του αγαπημένου φίλου Αστέριου Παρλαμέντα

Στα ενδότερα των patchers

Στο προηγούμενο τεύχος πραγματοποιήσαμε μια πρώτη γνωριμία με τους patchers. Σ' αυτό το τεύχος θα εμβαθύνουμε. Θα ξεκινήσουμε τη μελέτη μας από τη δομή των εκτελέσιμων αρχείων των Windows κι όταν κατανοήσουμε πώς είναι οργανωμένο το περιεχόμενό τους, θα αναλύσουμε μερικά τεχνάσματα που επιστρατεύουν οι ισχυροί patchers. Ακόμα κι αν δεν σκοπεύετε να φτιάξετε ποτέ κάποιο παρόμοιο εργαλείο, πιστεύουμε ότι όσα ακολουθούν θα σας δώσουν πολλές ενδιαφέρουσες ιδέες και θα σας καταστήσουν πιο πονηρούς – πάντα με την καλή έννοια.

Η συντριπτική πλειονότητα των αρχείων που περιέχουν εκτελέσιμο κώδικα στα Windows, ακολουθούν το πρότυπο PE32. Δεν αναφερόμαστε μόνο στα προγράμματα, αλλά και στις βιβλιοθήκες (DLLs). Το συγκεκριμένο πρότυπο προσδιορίζει με ακρίβεια τη θέση ενός συνόλου δεδομένων, τα οποία απαιτούνται για την εκτέλεση του κώδικα και το χειρισμό άλλων δεδομένων που ενδέχεται να ενσωματώνονται σε ένα αρχείο κώδικα. Αυτή η περιγραφή μπορεί να μη σας κατατοπίζει ιδιαίτερα για την ώρα, αλλά στην πορεία θα ξεκαθαρίσει το τοπίο. Το όνομα του προτύπου αποτελεί συντομογραφία του Portable Executable 32bit. Για την ιστορία, να πούμε ότι αποτελεί απόγονο του NE16 (New Executable 16bit) και πρόγονο του PE32+ (η εκδοχή για τα 64μπιτα εκτελέσιμα). Τα εκτελέσιμα PE32 εμφανίστηκαν για πρώτη φορά μαζί με τα Windows NT. Λειτουργούσαν ωστόσο και στα Windows 95, ενώ με τη βοήθεια του πακέτου Win32s εκτελούνταν ακόμη και στα Windows 3.1[1]. Την εποχή που εισήχθη το πρότυπο υπήρχαν ακόμα αρκετοί χρήστες του MS-DOS και ήταν πιθανό για ένα εκτελέσιμο PE32 να καταλήξει σ' ένα μηχάνημα με το παμπάλαιο λειτουργικό σύστημα της Microsoft. Τα συγκεκριμένα εκτελέσιμα, όμως, ήταν αδύνατο να τρέξουν στο MS-DOS. Ακριβώς γι' αυτό, τα αρχεία PE32 ξεκινούν με ένα header ειδικά για το MS-DOS (Real-Mode Header), αμέσως μετά το οποίο βρίσκεται μια περιοχή που φιλοξενεί οκτάμπιτο κώδικα (Real-Mode Stub). Όπως καταλαβαίνετε, ο οκτάμπιτος κώδικας προορίζεται επίσης για χρήση από το MS-DOS. Αναρωτιέστε ποια είναι η αποστολή του; Τυπώνει ένα μήνυμα και ενημερώνει το χρήστη για την αδυναμία εκτέλεσης του προγράμματος, στο περιβάλλον του MS-DOS.

```
00000000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..............
00000010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ........@.......
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00 00 00 00 00 D8 00 00 00  ................
00000040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ........!..L.!Th
00000050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00000060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS 
00000070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
00000080  26 C3 8E 85 62 A2 E0 D6 62 A2 E0 D6 62 A2 E0 D6  &...b...b...b...
00000090  E1 BE EE D6 6F A2 E0 D6 62 A2 E0 D6 6D A2 E0 D6  ....o...b...m...
```

Εικόνα 1
Με τη βοήθεια ενός hex editor, βλέπουμε ότι το header για το MS-DOS ξεκινά με τους χαρακτήρες MZ (bytes 0x5A4D). Πρόκειται για τα αρχικά του ονόματος Mark Zbikowski, ενός από τους αρχιτέκτονες του MS-DOS.

Τα πρώτα βήματα

Το MS-DOS έχει εξαλειφθεί προ πολλού (ακόμη κι από τα λογιστικά γραφεία). Επομένως, το Real-Mode Header που προοριζόταν για το MS-DOS έχει τεθεί πρακτικά στη διάθεση των ευρηματικών προγραμματιστών. Εφόσον τα σύγχρονα λειτουργικά δεν ασχολούνται μ' αυτή την περιοχή του αρχείου, θα μπορούσε κανείς να την τροποποιήσει ελεύθερα, χωρίς να παραπονεθεί κανένας μηχανισμός του συστήματος. Ακριβώς γι' αυτό, το Real-Mode Header χρησιμοποιήθηκε από πολύ νωρίς ως αποθηκευτικός χώρος και συμμετείχε σ' ένα από τα πρώτα συστήματα προστασίας των εφαρμογών. Σε γενικές γραμμές, οι προγραμματιστές αποθήκευαν εκεί το CRC32 του εκτελέσιμου κώδικα, ενώ φρόντιζαν να ενσωματώσουν στο πρόγραμμα και μερικές ρουτίνες που πραγματοποιούσουν τις ακόλουθες εργασίες:

- Άνοιγμα του ίδιου του εκτελέσιμου αρχείου (δηλαδή του εαυτού του).
- Δέσμευση μιας περιοχής μνήμης, ίσης με το μέγεθος του αρχείου.

1. Πόσοι από εσάς τα έχετε προλάβει;

- Ανάγνωση του εκτελέσιμου αρχείου και μεταφόρτωση στη μνήμη.
- Δημιουργία του CRC32, της περιοχής της μνήμης που περιέχει τον κώδικα.
- Ανάγνωση του αποθηκευμένου CRC32, από το MS-DOS header του εκτελέσιμου.
- Σύγκριση των δύο τιμών.

Αν είναι ίδιες, ο κώδικας της εφαρμογής δεν έχει τροποποιηθεί και η εκτέλεση προχωρά κανονικά. Αν τα δύο CRC32 είναι διαφορετικά, ο κώδικας της εφαρμογής έχει υποστεί τροποποιήσεις και η εκτέλεση τερματίζεται.

Ο μηχανισμός που περιγράψαμε προσαρτάτο αυτόματα στις διάφορες εφαρμογές, με τη βοήθεια ενός ειδικού εργαλείου. Αυτό το εργαλείο προσέθετε τις απαιτούμενες ρουτίνες στον κώδικα της εφαρμογής (αυτής που θέλαμε να προστατέψουμε), υπολόγιζε το CRC32 και το αποθήκευε σε μια κατάλληλη θέση εντός του MS-DOS header. Εν ολίγοις, το συγκεκριμένο εργαλείο αποτελούσε έναν πρόγονο των σύγχρονων patchers. Προφανώς, η προστασία με έναν απλό έλεγχο του CRC32 ξεπερνιέται πολύ εύκολα και δεν θα μπορούσε να εμποδίσει έναν cracker. Ήταν όμως ένα αποτελεσματικό μέτρο προστασίας, ενάντια στους προχωρημένους χρήστες που χρησιμοποιούσαν κάποιον hex editor για να βανδαλίζουν[2] τα προγράμματα.

Στοιχειώδης έλεγχος

Πριν ξεκινήσει την εργασία του ένας patcher οφείλει να αναλύσει τη δομή του προγράμματος που θέλουμε να προστατεύσουμε, για να αποφασίσει πόσες και ποιες επεμβάσεις μπορούν να πραγματοποιηθούν. Οι απαιτούμενες πληροφορίες βρίσκονται μέσα στο header του εκτελέσιμου αρχείου και η διάρθρωσή τους αποτελεί μέρος του προτύπου PE32. Τελικά, οι patchers ενσωματώνουν κι ένα μέρος της λειτουργικότητας των PE32 analyzers, των εργαλείων που αναλύουν τα εκτελέσιμα και παρουσιάζουν τα ευρήματά τους στο χρήστη.

```
00000000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..............
00000010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ........@.......
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00 00 00 00 00 D8 00 00 00  ................
00000040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ........!..L.!Th
00000050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00000060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS 
00000070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
00000080  26 C3 8E 85 62 A2 E0 D6 62 A2 E0 D6 62 A2 E0 D6  &...b...b...b...
00000090  E1 BE EE D6 6F A2 E0 D6 62 A2 E0 D6 6D A2 E0 D6  ....o...b...m...
000000A0  00 BD F3 D6 6D A2 E0 D6 62 A2 E1 D6 5A A3 E0 D6  ....m...b...Z...
000000B0  8A BD EB D6 54 A2 E0 D6 8A BD EA D6 47 A2 E0 D6  ....T.......G...
000000C0  DA A4 E6 D6 63 A2 E0 D6 52 69 63 68 62 A2 E0 D6  ....c...Richb...
000000D0  00 00 00 00 00 00 00 00 50 45 00 00 4C 01 03 00  ........PE..L...
```

Εικόνα 2
Στη διεύθυνση 0x3C του MS-DOS header υπάρχει πάντα η διεύθυνση εκκίνησης του PE header. Αυτό το header, με τη σειρά του, ξεκινά πάντα με τους χαρακτήρες PE00. Αν τους συναντήσουμε, επιβεβαιώνουμε ότι βρισκόμαστε στην αρχή ενός PE32 header.

2. Μερικές κλασικές επεμβάσεις αποτελούσαν η αλλαγή του copyright και του ονόματος του προγραμματιστή, όπως και η αλλοίωση των μηνυμάτων που εμφάνιζε το πρόγραμμα κατά τη λειτουργία του.

Ξεκινώντας τη γνωριμία μας με τη δομή των PE32, θα αναφέρουμε τα στοιχεία εκείνα που ταυτοποιούν ένα εκτελέσιμο αρχείο ως εκτελέσιμο τύπου PE32. Για να πραγματοποιήσουμε αυτόν τον έλεγχο, πρέπει να εντοπίσουμε μερικές χαρακτηριστικές τιμές, σε συγκεκριμένες θέσεις του MS-DOS header αλλά και του PE header. Το MS-DOS header βρίσκεται στην αρχή των εκτελέσιμων αρχείων και ξεκινάει πάντα με τους χαρακτήρες MZ[3]. Έτσι, αν συναντήσουμε τα σχετικά bytes στο ξεκίνημα ενός αρχείου, έχουμε μια πρώτη ένδειξη ότι περιλαμβάνει εκτελέσιμο κώδικα. Στη συνέχεια μπορούμε να αγνοήσουμε τα υπόλοιπα δεδομένα του MS-DOS header, μέχρι να φτάσουμε στη διεύθυνση 0x3C[4]. Εκεί βρίσκεται ένα DWORD που περιλαμβάνει τη διεύθυνση εκκίνησης του PE32 header. Αυτό το header ξεκινάει πάντα με τους χαρακτήρες PE00. Επομένως, αν τους συναντήσουμε στη διεύθυνση που διαβάσαμε από τη θέση 0x3C του MS-DOS header, συμπεραίνουμε ότι εκεί ξεκινά όντως ένα PE32 header. Στη συνέχεια παραθέτουμε ένα απόσπασμα από τη δομή δεδομένων (struct), που περιγράφει τα περιεχόμενα του PE32 header:

```
typedef struct
{
      DWORD PESig;
      WORD  CPUType;
      WORD  ObjectEntries;
      DWORD TimeDate;
      DWORD SymbolPointer;
      DWORD SymbolEntries;
      WORD  SizeOfOptHeader;
      WORD  Flags;
      WORD  Magic;
      WORD  LinkVersion;
      DWORD SizeOfCode;
      DWORD InitDataSize;
      DWORD UnInitDataSize;
      DWORD EntryPoint;
      DWORD BaseOfCode;
      DWORD BaseOfData;
      DWORD ImageBase;
      DWORD SectionAlignment;
      DWORD FileAlignment;
      DWORD OSVersion;
```

3. Πρόκειται για τα αρχικά του ονόματος Mark Zbikowski — ενός από τους αρχιτέκτονες του MS-DOS.

4. Αν θέλετε να δείτε τη διάρθρωση των δεδομένων που περιλαμβάνει το MS-DOS header, μπορείτε να ανατρέξετε στο αρχείο winnt.h και να ψάξετε για τη δομή (struct) με όνομα _IMAGE_DOS_HEADER. Τα περιεχόμενα του winnt.h μπορείτε να τα βρείτε εύκολα και στο Internet (π.χ., στη διεύθυνση http://source.winehq.org/source/include/winnt.h).

```
DWORD ImageVersion;
DWORD SubSysVersion;
DWORD Reserved4;
DWORD ImageSize;
DWORD HeaderSize;
DWORD CheckSum;
WORD  SubSystemFlags;
WORD  DllFlags;
DWORD StackReserve;
DWORD StackCommit;
DWORD HeapReserve;
DWORD HeapCommit;
DWORD LoaderFlags;
DWORD DataDirectorySize;
DWORD ExportTable;
DWORD ExportSize;
DWORD ImportTable;
DWORD ImportSize;
DWORD ResourceTable;
DWORD ResourceSize;
DWORD ExceptionTable;
DWORD ExceptionSize;
DWORD CertificateTable;
DWORD CertificateSize;
DWORD RelocTable;
DWORD RelocSize;
DWORD DebugTable;
DWORD DebugSize;
DWORD ArchitectureTable;
DWORD ArchitectureSize;
DWORD GlobalPTRTable;
DWORD GlobalPTRSize;
DWORD TLSTable;
DWORD TLSSize;
DWORD LoadConfigTable;
```

```
        DWORD LoadConfigSize;
        DWORD BoundImportTable;
        DWORD BoundImportSize;
        DWORD ImportAddressTable;
        DWORD ImportAddressSize;
        DWORD DelayImportTable;
        DWORD DelayImportTableSize;
        DWORD CLRRuntimeHeader;
        DWORD CLRRuntimeHeaderSize;
        DWORD Reserved1;
        DWORD Reserved2;
    } PE32Header;
```

Οι τιμές που έχουμε μαρκάρει παρουσιάζουν ιδιαίτερο ενδιαφέρον και θα μας απασχολήσουν στην πορεία του άρθρου. Ας επιστρέψουμε όμως στο θέμα μας. Για να σιγουρευτούμε ότι το αρχείο που εξετάζουμε είναι όντως εκτελέσιμο, πρέπει να τσεκάρουμε ορισμένες τιμές εντός του PE32 header. Συγκεκριμένα, πρέπει να επιβεβαιώσουμε ότι το πεδίο PEHeader32 → Magic έχει την τιμή 0x10B ή την 0x20B. Επιπρόσθετα, τα πεδία PEHeader32 → CPUType και PEHeader32 → SubSystemFlags πρέπει να έχουν την τιμή μηδέν.

Περισσότερα στοιχεία

Ανάμεσα σε άλλες σημαντικές πληροφορίες που περιλαμβάνει το PE32 header, αναφέρεται και το πλήθος των ξεχωριστών τμημάτων (sections) που απαρτίζουν το εκτελέσιμο αρχείο. Τα εν λόγω τμήματα αποτελούν διακριτές περιοχές (αυτοτελείς ενότητες) του αρχείου, στις οποίες μπορεί να περιλαμβάνεται κώδικας ή δεδομένα. Το συνολικό πλήθος των sections δηλώνεται στην αρχή του header, με το μέγεθος ObjectEntries. Ωστόσο, τα επιμέρους χαρακτηριστικά για κάθε section αναφέρονται σε έναν ξεχωριστό πίνακα, ο οποίος βρίσκεται στο τέλος του header. Παρακάτω θα δείτε μια ρουτίνα από ένα παλιότερο (δικό μας) πρόγραμμα, η οποία διαβάζει τον πίνακα των sections και εμφανίζει τα περιεχόμενά του:

```
    sprintf(tmp, "header.ObjectEntries = %d;", FileID,myLocalFile.
    PEHeader32->ObjectEntries);
    LogIt(tmp);
    (DWORD)mySection = (DWORD)myLocalFile.PEHeader32 + sizeof(PE32Header);
    for (i=0; i<myLocalFile.PEHeader32->ObjectEntries; i++)
    {
```

```
        sprintf(tmp, "Found '%s' file offset:0x%X (size=0x%X), Virtual
        offset:0x%X (size=0x%X), Flags:0x%X", mySection[i].ObjectName,
        mySection[i].PObjOffset, mySection[i].PObjSize, mySection[i].RVA,
        mySection[i].VObjSize, mySection[i].Characteristics);
        LogIt(tmp);
    }
```

Η ρουτίνα περιλαμβάνει ένα loop, το οποίο επαναλαμβάνεται τόσες φορές όσα είναι και τα sections του αρχείου. Σε κάθε επανάληψη του βρόχου το πρόγραμμα τυπώνει μια σειρά πληροφοριών για το εκάστοτε section, όπως αυτές:

```
name: .text
VirtualSize: 00005AFA
VirtualOffset: 00001000
PSize: 00005C00
POffset: 00000400
flags: 60000020
```

Το όνομα αποτελεί μια συμβολοσειρά που δηλώνει αν το συγκεκριμένο τμήμα περιλαμβάνει κώδικα ή δεδομένα. Στο παράδειγμα, η τιμή του ονόματος είναι ".text" και πρόκειται για section που περιέχει κώδικα. Σημειώστε ότι διαφορετικοί compilers χρησιμοποιούν ελαφρώς διαφορετικά ονόματα για τον τύπο των sections. Το VirtualSize δηλώνει το μέγεθος που καταλαμβάνει το section όταν φορτώνεται στη μνήμη, ενώ το VirtualOffset αποτελεί τη σχετική[5] διεύθυνση μνήμης στην οποία ξεκινούν τα περιεχόμενα του section. Το PSize αποτελεί το μέγεθος που καταλαμβάνει το συγκεκριμένο section εντός του εκτελέσιμου αρχείου, ενώ το POffset δείχνει τη σχετική διεύθυνση[6] στην οποία ξεκινάει το section, μέσα στο αρχείο. Το τελευταίο μέγεθος (flags) περιλαμβάνει μια σειρά βασικών ιδιοτήτων του section. Για παράδειγμα, το αν είναι εγγράψιμο (write), το αν είναι αναγνώσιμο (read), το αν είναι εκτελέσιμο (executable) κ.ο.κ.

Αν κατασκευάσετε τον δικό σας patcher ή κάποιο άλλο πρόγραμμα που τροποποιεί εκτελέσιμα αρχεία, θα πρέπει να τσεκάρετε πάντα τις ιδιότητες του section που προτίθεστε να αλλοιώσετε. Συγκεκριμένα, πρέπει να ελέγχετε αν είναι εγγράψιμο κι αν δεν είναι να το καθιστάτε εσείς, τροποποιώντας κατάλληλα τις ρυθμίσεις του. Σε διαφορετική περίπτωση, αν προσπαθήσετε να αλλάξετε έστω κι ένα byte από τα περιεχόμενά του θα πάρετε ένα ωραιότατο "Access Violation Error". Αν θέλετε να δείτε αναλυτικά τις ρυθμίσεις που επιδέχονται τα sections, ανατρέξτε στο κεφάλαιο 4.1 (Section Flags) του κειμένου "Microsoft Portable Executable and Common Object File Format Specification". Μπορείτε να το κατεβάσετε από το http://bit.ly/dh033pecoff.

5. Ως προς τη διεύθυνση στην οποία ξεκινά η περιοχή μνήμης που χρησιμοποιεί το πρόγραμμα.
6. Ως προς το σημείο του αρχείου που ξεκινά ο κώδικας.

Αλλαγή "συστήματος αναφοράς"

Πριν προχωρήσουμε πρέπει να σταθούμε σε μια ενδιαφέρουσα λεπτομέρεια: Στο παράδειγμα που δώσαμε παραπάνω, η έκταση που καταλαμβάνει το section μέσα στο αρχείο είναι ελαφρώς μεγαλύτερη (5C00 bytes) από την αντίστοιχη εντός της μνήμης (5AFA bytes). Αυτή η διαφορά οφείλεται στο γεγονός ότι τα δεδομένα του εκάστοτε section είναι οργανωμένα σε "πακέτα" συγκεκριμένου μεγέθους. Με άλλα λόγια, τα δεδομένα είναι "ευθυγραμμισμένα" (aligned) ανά συγκεκριμένο πλήθος bytes. Αυτές οι ποσότητες δεν είναι ίδιες για τη μνήμη και για το αρχείο και, φυσικά, οι σχετικές τιμές αναφέρονται στο PE32 header. Πρόκειται για τα μεγέθη FileAlignment (μέγεθος ευθυγράμμισης στο αρχείο) και SectionAlignment (μέγεθος ευθυγράμμισης στη μνήμη). Αν σας βοηθάει, θα μπορούσατε να σκεφτόσαστε τα συγκεκριμένα μεγέθη ως το λεγόμενο allocation unit (μονάδα εκχώρησης) των δίσκων.

Μελετώντας το ίδιο παράδειγμα, είναι βέβαιο ότι εντοπίσατε και μία ακόμα διαφοροποίηση: Η διεύθυνση ενός section στη μνήμη δεν ταυτίζεται με αυτή στο αρχείο. Αναφέραμε μάλιστα ότι αυτές οι διευθύνσεις είναι σχετικές. Αναρωτιέστε πού θα βρούμε τα αντίστοιχα σημεία αναφοράς; Η απάντηση δίνεται πάλι μέσα στο PE32 header. Το μέγεθος ImageBase δηλώνει τη διεύθυνση μνήμης στην οποία φορτώνεται το αρχείο, ενώ το μέγεθος BaseOfCode δηλώνει τη διεύθυνση στην οποία ξεκινάει ο κώδικας εντός του αρχείου.

Ένα πρόγραμμα που πραγματοποιεί επεμβάσεις σε εκτελέσιμα αρχεία (ένας patcher, για παράδειγμα), θα χρειαστεί αρκετές φορές να αλλάξει το μέγεθος ενός section, να μετακινήσει κάποιο άλλο, να ανατρέξει σε μια συγκεκριμένη θέση ενός τρίτου κ.λπ. Για να πραγματοποιηθούν σωστά αυτές οι εργασίες το πρόγραμμά μας πρέπει να χειρίζεται με άνεση τις διευθύνσεις που αναφέραμε παραπάνω και να είναι σε θέση να τις μετατρέπει από το ένα είδος στο άλλο. Γι' αυτό το σκοπό έχουμε γράψει δύο μικρές ρουτίνες, που μετατρέπουν τις σχετικές διευθύνσεις μνήμης στις αντίστοιχες διευθύνσεις εντός του αρχείου και το αντίστροφο. Πρόκειται για τις ρουτίνες RVA2POffset και POffset2RVA[7], που μπορείτε να δείτε παρακάτω:

```
DWORD POffset2RVA(DWORD GivenOffset)
{
     DWORD i;
     (DWORD)mySection = (DWORD)myLocalFile.PEHeader32 + sizeof(PE32Header);
     for (i = 0; i <myLocalFile.PEHeader32->ObjectEntries; i++)
     {
        if ( (mySection[i].PObjOffset <= GivenOffset) &&
((mySection[i].PObjOffset + mySection[i].PObjSize) >= GivenOffset) )
        {
           return (GivenOffset - mySection[i].PObjOffset) + mySection[i].RVA;
        }
     }
```

7. RVA είναι η σχετική διεύθυνση στη μνήμη (Relative Virtual Address) και POffset είναι η σχετική διεύθυνση εντός του αρχείου.

```
        return 0;
    }

    DWORD RVA2POffset(DWORD GivenRVA)
    {
        DWORD i;
        (DWORD)mySection = (DWORD)myLocalFile.PEHeader32 +
    sizeof(PE32Header);
        for (i = 0; i <myLocalFile.PEHeader32->ObjectEntries; i++)
        {
           if ((mySection[i].RVA <= GivenRVA) && ((mySection[i].RVA +
           mySection[i].VObjSize) >= GivenRVA))
           {
              return (GivenRVA - mySection[i].RVA) + mySection[i].
    PObjOffset;
           }
        }
    }
```

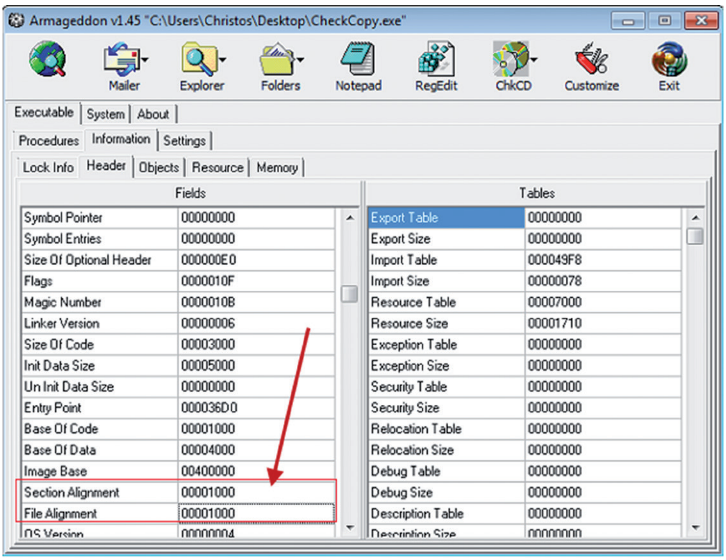
Εξοικονόμηση χώρου

Σ' αυτό το σημείο θα πραγματοποιήσουμε μια σύντομη αλλά ενδιαφέρουσα παρέκκλιση: Θα αφήσουμε για λίγο τους patchers και θα αναφερθούμε στον τρόπο λειτουργίας κάποιων εργαλείων του παρελθόντος. Θα μιλήσουμε για προγράμματα που δεν προσέθεταν μηχανισμούς προστασίας στα εκτελέσιμα αρχεία, αλλά μπορούσαν να συρρικνώσουν το μέγεθός τους. Στο παρελθόν, βλέπετε, τα προγράμματα μεταφέρονταν σε δισκέτες των οποίων το μέγεθος ήταν πολύ μικρό. Δεν μιλάμε για τις πρώτες δισκέτες των 5,25 ιντσών, που χωρούσαν μόλις 360 KB, γιατί εκείνη την περίοδο δεν υπήρχε το PE32. Μιλάμε όμως για τα τέλη της εποχής του MS-DOS, που τα εκτελέσιμα PE32 είχαν ήδη εμφανιστεί και οι τότε δισκέτες (3,5 ιντσών) χωρούσαν 1,44MB. Όπως καταλαβαίνετε, η ανάγκη να συμπιέζουμε τα προγράμματα ήταν αυξημένη. Ένα εργαλείο της εποχής που συμπίεζε τα εκτελέσιμα αρχεία ήταν το δημοφιλές pklite. Ουσιαστικά, λειτουργούσε σαν ένας απλός patcher[8] κι εργαζόταν ως εξής: Εντόπιζε τα sections που θα μπορούσαν να συμπιεστούν, τα συμπίεζε κι ενσωμάτωνε στο πρόγραμμα την κατάλληλη ρουτίνα αποσυμπίεσης.

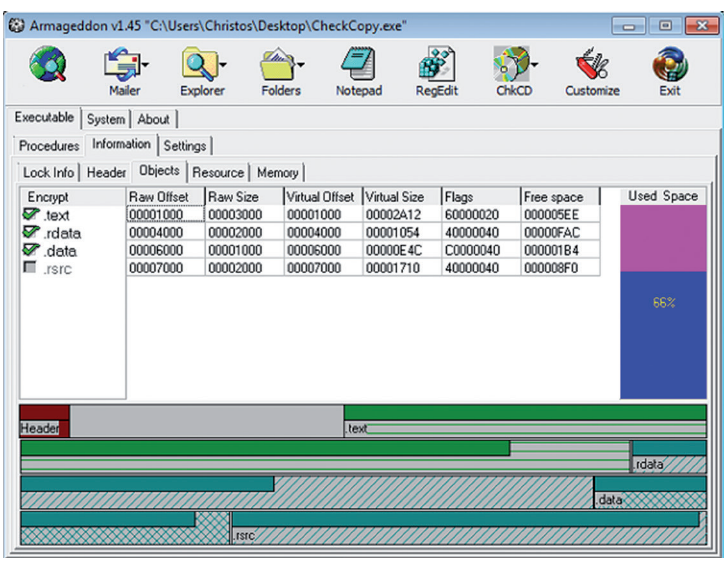
Θα μπορούσαμε άραγε να πετύχουμε κάτι παρόμοιο; Η απάντηση είναι προφανώς καταφατική! Θα παρουσιάσουμε ωστόσο μια εναλλακτική μέθοδο, πιο ενδιαφέρουσα και πιο πονηρή. Θυμηθείτε για λίγο τη διαφοροποίηση που παρουσιάζει η έκταση ενός section στη μνήμη κι εντός του αρχείου. Όπως είπαμε, η διαφορά αυτή οφείλε-

8. Τελικά δεν ξεφεύγουμε εντελώς από το θέμα ;)

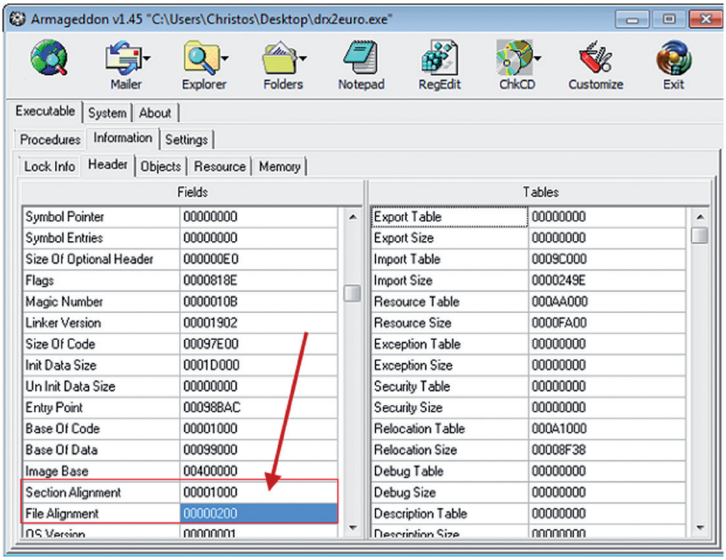
ται στο alignment. Ας υποθέσουμε, λοιπόν, ότι ένα section αποτελείται από 0x6415 bytes κώδικα. Αν το FileAlignment οριστεί στα 0x1000 bytes, όταν αποθηκευτεί το συγκεκριμένο section στο αρχείο θα δεσμευτούν 0x7000 bytes. Βλέπετε, τα 0x7000 bytes αποτελούν το μικρότερο ακέραιο πολλαπλάσιο των 0x1000 bytes, στο οποίο χωράει ο κώδικας του section. Αν όμως το FileAlignment οριστεί στα 0x200 bytes, το ίδιο section θα καταλάβει 0x6600 bytes στο αρχείο. Με λίγα λόγια, επιλέγοντας μικρότερη τιμή για το FileAlignment καταφέραμε να εξοικονομήσουμε χώρο! Το κέρδος που πετυχαίνουμε με αυτό το κόλπο δεν είναι σταθερό για όλα τα εκτελέσιμα. Επηρεάζεται κάθε φορά από το πλήθος των sections, το μέγεθος του καθενός και την αρχική τιμή του FileAlignment.



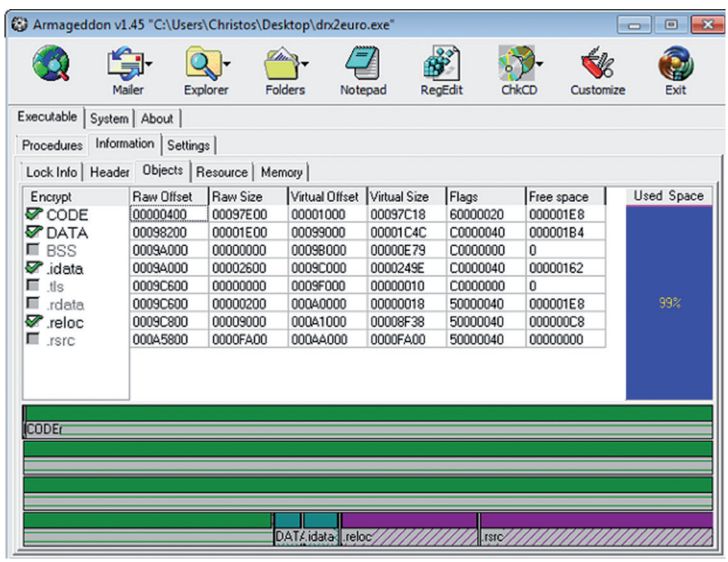
Εικόνα 3
Στην εικόνα βλέπουμε τα χαρακτηριστικά ενός εκτελέσιμου, όπως παρουσιάζονται μέσα από ένα δικό μας εργαλείο. Το εκτελέσιμο που έχουμε ανοίξει γράφτηκε στη Visual C v.6. και ο compiler όρισε το FileAlignment στα 0x1000bytes.



Εικόνα 4
Το ίδιο εργαλείο παρουσιάζει και το χώρο που "χάνεται" εξαιτίας του επιλεγμένου μεγέθους του FileAlignment. Η θεωρητική σπατάλη χώρου ανέρχεται στο 34%.



Εικόνα 5
Το εκτελέσιμο που έχουμε να εξετάσουμε εδώ δημιουργήθηκε από τον compiler του Delphi. Ο συγκεκριμένος compiler φάνηκε πιο φειδωλός με το χώρο κι όρισε το FileAlignment στα 0x200 bytes.



Εικόνα 6
Εξαιτίας της μικρής τιμής του FileAlignment, η θεωρητική σπατάλη χώρου αγγίζει μόλις το 1%!

Ακολουθεί μια ρουτίνα που υλοποιεί το παραπάνω τέχνασμα. Όπως θα διαπιστώσετε εξετάζοντας τον κώδικα, η ρουτίνα μας θέτει την τιμή του FileAlignment στα 0x200 bytes. Αυτό όμως δεν είναι αρκετό για να εξοικονομήσουμε χώρο. Η αλλαγή του FileAlignment δημιουργεί "φυσαλίδες" ανάμεσα στα sections, τις οποίες πρέπει να εκμεταλλευτούμε. Ο τρόπος για να πετύχουμε κάτι τέτοιο είναι να μετακινήσουμε όλα τα sections και να τα φέρουμε πιο κοντά το ένα με το άλλο. Έτσι, μετά την αλλαγή του FileAlignment, η ρουτίνα μας πραγματοποιεί και τις κατάλληλες μετακινήσεις...

```
DWORD ReconstructSections (void)
{
     int i;
     div_t div_result;
     DWORD tmpSize;
     char *tmpBuf;
     LogIt("Reconstructing sections...");
     (DWORD)mySection = (DWORD)myLocalFile.PEHeader32 +
     sizeof(PE32Header);
     myLocalFile.PEHeader32->FileAlignment = 0x200;
     for (i=0; i<myLocalFile.PEHeader32->ObjectEntries; i++)
     {
        if ( mySection[i].VObjSize < mySection[i].PObjSize )
        {
           div_result = div( mySection[i].VObjSize, myLocalFile.
           PEHeader32->FileAlignment );
           if ( div_result.rem > 0 )
           {
              if (mySection[i].PObjSize != (div_result.quot + 1) *
              myLocalFile.PEHeader32->FileAlignment)
              {
                 sprintf(tmp, "Changing '%s' from 0x%08X to 0x%08X",
                 mySection[i].ObjectName, mySection[i].PObjSize, (div_
                 result.quot + 1) * myLocalFile.PEHeader32
                 ->FileAlignment);
                 LogIt(tmp);
                 mySection[i].PObjSize = (div_result.quot + 1) *
                 myLocalFile.PEHeader32->FileAlignment;
              }
           }
        }
     }
     myLocalFile.FreeSpace = mySection[myLocalFile.PEHeader32
     ->ObjectEntries - 1].PObjOffset;

     tmpBuf = (char*)((DWORD)myLocalFile.Buffer);
     for (tmpSize=mySection[0].PObjOffset-1; tmpSize>0x400; tmpSize–)
     {
        if ( tmpBuf[tmpSize] != 0 )
        {
```

```
            break;
        }
    }
    div_result = div( tmpSize, myLocalFile.PEHeader32->FileAlignment );
    if ( div_result.rem > 0 )
    {
        tmpSize = (div_result.quot + 1) * myLocalFile.PEHeader32
        ->FileAlignment;
    }
    if ( tmpSize != mySection[0].PObjOffset )
    {
        sprintf(tmp, "Moving '%s' from 0x%08X to 0x%08X",
        mySection[0].ObjectName, mySection[0].PObjOffset, tmpSize);
        LogIt(tmp);
        memcpy((char*)((DWORD)myLocalFile.Buffer + tmpSize),
              (char*)((DWORD)myLocalFile.Buffer + mySection[0].
              PObjOffset),
              mySection[0].PObjSize);
        mySection[0].PObjOffset = tmpSize;
    }

    for (i=0; i < myLocalFile.PEHeader32->ObjectEntries - 1; i++)
    {
        tmpSize = mySection[i].PObjOffset + mySection[i].PObjSize;
        div_result = div( tmpSize, myLocalFile.PEHeader32
        ->FileAlignment );
        if ( div_result.rem > 0 )
        {
            tmpSize = (div_result.quot + 1) * myLocalFile.PEHeader32
            ->FileAlignment;
        }
        if ( tmpSize < mySection[i+1].PObjOffset )
        {
            sprintf(tmp, "Moving '%s' from 0x%08X to 0x%08X",
            mySection[i+1].ObjectName, mySection[i+1].PObjOffset,
            tmpSize);
            LogIt(tmp);
            memcpy((char*)((DWORD)myLocalFile.Buffer + tmpSize),
                  (char*)((DWORD)myLocalFile.Buffer + mySection[i+1].
                  PObjOffset),
```

```
                    mySection[i+1].PObjSize);
                mySection[i+1].PObjOffset = tmpSize;
            }
        }
        myLocalFile.FreeSpace -= mySection[myLocalFile.PEHeader32
        ->ObjectEntries - 1].PObjOffset;

        if ( myLocalFile.FreeSpace > 0 )
        {
            sprintf(tmp, "Released %d bytes", myLocalFile.FreeSpace);
            LogIt(tmp);
        }
        return 0;
    }
```

Αν ενσωματώσετε τον κώδικά μας σε κάποιο δικό σας πρόγραμμα και καταφέρετε να τροποποιήσετε με επιτυχία το μέγεθος ενός εκτελέσιμου, αξίζετε συγχαρητήρια! Θα σας προτείναμε μάλιστα να επικοινωνήσετε μαζί μας, για να μοιραστούμε ιδέες και απόψεις.

Εξοικονόμηση και όχι μόνο

Μιας και ανοίξαμε το θέμα της εξοικονόμησης χώρου, μπορούμε να συνεχίσουμε και να το συνδυάσουμε με τη βασική αποστολή ενός patcher: Την παρεμπόδιση της δουλειάς ενός cracker. Για να πετύχουμε αυτό το διπλό κέδρος, της συρρίκνωσης του προγράμματος και της αύξησης της ασφάλειάς του, αρκεί να καταργήσουμε τους πίνακες debug table και relocation table. Η επιλογή του ρήματος "καταργώ" δεν ήταν τυχαία, αφού στην πραγματικότητα δεν θα διαγράψουμε τους δύο πίνακες, αλλά θα αλλάξουμε τη χρήση τους. Το τι ακριβώς θα κάνουμε θα το δούμε παρακάτω. Για την ώρα θα μάθουμε τι περιλαμβάνει ο καθένας.

Το debug table δημιουργείται από τον compiler και προστίθεται στο εκάστοτε εκτελέσιμο αυτόματα. Όπως δηλώνει και το όνομά του, περιέχει πληροφορίες που βοηθάνε στον εντοπισμό σφαλμάτων. Οι πληροφορίες αυτές αποκαλύπτουν διάφορες λεπτομέρειες της λειτουργίας του προγράμματος και, συνεπώς, αποτελούν πολύτιμο εφόδιο για έναν cracker. Ο δημιουργός ενός προγράμματος μπορεί να απενεργοποιήσει την προσθήκη του debug table, εκτελώντας τον compiler με την κατάλληλη παράμετρο. Δεν αποκλείεται όμως και να το ξεχάσει. Από την άλλη, το relocation table περιγράφει το πώς πρέπει να τροποποιηθούν τα δεδομένα ενός section, όταν το εκτελέσιμο τοποθετείται σε διαφορετική θέση μνήμης από αυτήν που έχει ορίσει ο compiler.

Έχοντας κατά νου την αποστολή των δύο πινάκων, μπορούμε να προχωρήσουμε στον κατάλληλο χειρισμό του καθενός. Μια πρώτη σκέψη θα ήταν να διαγράψουμε το debug table. Αν μάλιστα βρίσκεται μέσα σε κάποιο ξεχωριστό section, θα μπορού-

σαμε να το διαγράψουμε ολόκληρο για να ελευθερώσουμε χώρο. Ο σωστός τρόπος για τη διαγραφή του πίνακα θα ήταν να τον γεμίσουμε με μηδενικά (ή με άλλα τυχαία δεδομένα) και στη συνέχεια να ορίσουμε το μέγεθός του ίσο με μηδέν. Επιπρόσθετα, μπορούμε να "καταστρέψουμε" και το δείκτη που παραπέμπει στην αρχή του πίνακα. Για να μηδενίσουμε το μέγεθος του πίνακα, αρκεί να επέμβουμε στο PE32 header του εκτελέσιμου και να μηδενίσουμε το μέγεθος DebugSize. Για την "καταστροφή" του σχετικού δείκτη, αρκεί να επέμβουμε πάλι στο header και να δώσουμε μια τυχαία τιμή στο μέγεθος DebugTable. Σημειώστε, πάντως, ότι θα μπορούσαμε να μη διαγράψουμε τον εν λόγω πίνακα, αλλά να τον αξιοποιήσουμε ως αποθηκευτικό χώρο. Για παράδειγμα, θα μπορούσαμε να ορίσουμε το section που τον περιλαμβάνει ως εκτελέσιμο και στη συνέχεια να τοποθετήσουμε εκεί τον κώδικα που εισάγει ο patcher ;)

Στην περίπτωση του relocation table τα πράγματα δεν είναι τόσο απλά. Κατ' αρχάς, σημειώστε ότι ο συγκεκριμένος πίνακας είναι άχρηστος για τα εκτελέσιμα αρχεία EXE. Ο κώδικας των προγραμμάτων φορτώνεται πάντα σε μια σταθερή διεύθυνση (0x400000) και ποτέ δεν εκτελούνται δύο προγράμματα μέσα στο ίδιο process space. Στις δυναμικές βιβλιοθήκες, αντίθετα, το relocation table είναι απολύτως απαραίτητο. Τα προγράμματα δεν φορτώνουν ποτέ μία μόνο βιβλιοθήκη. Επομένως, στο ίδιο process space φορτώνονται αρκετές βιβλιοθήκες και, προφανώς, η τοποθέτησή τους στην ίδια διεύθυνση δεν είναι εφικτή. Επομένως, αν το αρχείο που εξετάζουμε είναι EXE, μπορούμε να μεταχειριστούμε το relocation table με τον τρόπο που χειριστήκαμε το debug table. Αν όμως πρόκειται για κάποια βιβλιοθήκη, πρέπει να το αφήσουμε ανέπαφο.

Επιστροφή στο σκληρό πυρήνα (όχι του Ευρώ)

Ο τρόπος αντιμετώπισης του relocation table που παρουσιάσαμε προηγουμένως ήταν μάλλον αφελής. Μέχρι πριν λίγο, βλέπετε, είχαμε εστιάσει στην εξοικονόμηση χώρου και είχαμε απομακρυνθεί από την πραγματική αποστολή των patchers. Αν θέλουμε να αυξήσουμε πραγματικά την ασφάλεια ενός "κλειδωμένου" εκτελέσιμου, μπορούμε να αξιοποιήσουμε την ύπαρξη του relocation table με πιο έξυπνους τρόπους. Ας αναφέρουμε δύο από αυτούς.

- Στην περίπτωση μιας βιβλιοθήκης αντιγράφουμε τα περιεχόμενα του relocation table σε κάποια άλλη θέση, αλλάζοντας ταυτόχρονα τη δομή τους για να γίνουν αγνώριστα! Στη συνέχεια εξασφαλίζουμε ότι ο patcher θα εισάγει τον κατάλληλο κώδικα στο πρόγραμμα που καλεί τη βιβλιοθήκη, ο οποίος θα επαναφέρει το relocation table στην κανονική του μορφή. Έτσι, όταν εκτελείται το πρόγραμμα που χρησιμοποιεί την κλειδωμένη βιβλιοθήκη, το relocation table θα διορθώνεται αυτόματα. Αναρωτιέστε τι κερδίζουμε με όλο αυτό; Πρέπει να λάβετε υπόψη σας ότι κυκλοφορούν διάφορα εργαλεία που ειδικεύονται στο ξήλωμα των patchers και στην επαναφορά των προστατευμένων προγραμμάτων στην αρχική τους μορφή. Με το κολπάκι που περιγράψαμε ένα τέτοιο εργαλείο θα αποτύχει να "διορθώσει" την κλειδωμένη βιβλιοθήκη, διότι δεν θα βρει πουθενά τα περιεχόμενα του relocation table!
- Προαιρετικά, μπορούμε να κάνουμε και κάτι πιο πονηρό. Αφού μετακινήσουμε τα πραγματικά περιεχόμενα του relocation table σε κάποια άλλη θέση (αλλάζοντας και τον τρόπο οργάνωσής τους για να είναι αγνώριστα) μπορούμε να βάλουμε στη θέση τους ψεύτικα δεδομένα (fake data). Έτσι, ένα εργαλείο που

επαναφέρει τις κλειδωμένες βιβλιοθήκες στην αρχική τους μορφή, θα ανασκευάσει το relocation table με τα ψεύτικα δεδομένα και θα δημιουργήσει μια βιβλιοθήκη που δεν θα λειτουργεί σωστά.

Τα κολπάκια που μπορούμε να κάνουμε με το relocation table, εφόσον υπάρχει σε κάποιο εκτελέσιμο και απαιτείται για τη λειτουργία του, είναι πάρα πολλά. Ένα απλούστερο από αυτά που αναφέραμε θα ήταν να κρυπτογραφήσουμε τα δεδομένα του και να φροντίσουμε για τη δυναμική αποκρυπτογράφησή τους, κατά την εκτέλεση του προγράμματος. Με αυτόν τον τρόπο θα καταφέρναμε πάλι να αποκρούσουμε τις προσπάθειες αφαίρεσης του εμφυτευμένου κώδικα.

Η χρησιμότητα του import table

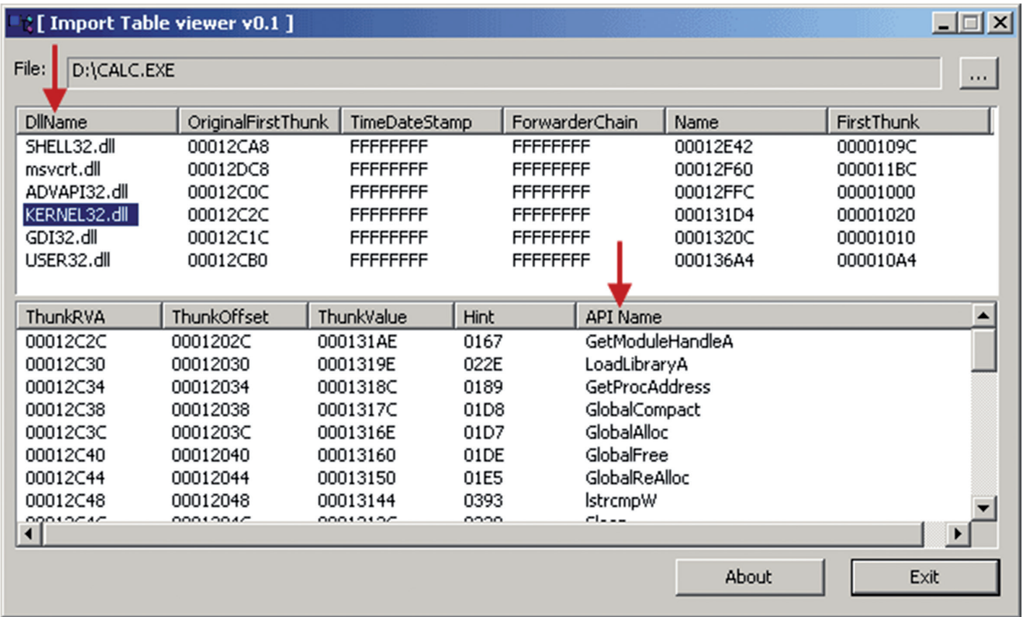
Στο προηγούμενο τεύχος, στο εισαγωγικό μας άρθρο για τους patchers, αναφερθήκαμε στην τεράστια σημασία του import table. Σε αυτόν τον πίνακα αναφέρονται όλες οι βιβλιοθήκες που απαιτεί ένα πρόγραμμα, καθώς και οι συναρτήσεις που χρησιμοποιούνται από κάθε βιβλιοθήκη. Θέλετε να μάθετε γιατί είναι αναγκαίος αυτός ο πίνακας;

Αν γράψουμε ένα απλό πρόγραμμα, το οποίο εμφανίζει ένα παράθυρο διαλόγου (MessageBox) και μετά κλείνει, χωρίς να το καταλάβουμε θα έχουμε χρησιμοποιήσει δύο συναρτήσεις από βιβλιοθήκες του συστήματος. Η πρώτη συνάρτηση είναι η MessageBoxA κι ανήκει στη βιβλιοθήκη User32.DLL. Η δεύτερη συνάρτηση είναι η ExitProcess και βρίσκεται στη βιβλιοθήκη Kernel32.DLL. Όμως ο compiler που θα μεταγλωττίσει το πρόγραμμά μας δεν μπορεί να γνωρίζει σε ποια θέση μνήμης θα φορτωθούν οι βιβλιοθήκες. Κατ' επέκταση, δεν γνωρίζει ούτε και τη διεύθυνση στην οποία θα ξεκινά ο κώδικας των ζητούμενων συναρτήσεων. Γι' αυτό το λόγο δημιουργεί τον πίνακα import table, ο οποίος περιέχει μία εγγραφή για κάθε βιβλιοθήκη και για κάθε συνάρτηση που απαιτεί ο κώδικάς μας. Σε κάθε εγγραφή προβλέπεται χώρος και για τη συμπλήρωση της αντίστοιχης διεύθυνσης, αλλά ο compiler δεν συμπληρώνει τίποτα. Όταν εκτελούμε το πρόγραμμα, το λειτουργικό σύστημα φορτώνει τις βιβλιοθήκες που αναφέρονται στο import table, εντοπίζει και τις απαιτούμενες συναρτήσεις και φροντίζει να συμπληρώσει το import table με τις κατάλληλες διευθύνσεις.

Η χρήση του import table δεν αρκεί για να λυθεί το πρόβλημα με τις άγνωστες διευθύνσεις των βιβλιοθηκών και των συναρτήσεων. Ο compiler λαμβάνει κι ένα ακόμα μέτρο: Σε οποιοδήποτε σημείο του κώδικα πραγματοποιείται κλήση προς τη συνάρτηση μιας βιβλιοθήκης, δεν χρησιμοποιείται η εντολή που πραγματοποιεί "direct call" (απευθείας κλήση της διεύθυνσης που δίνουμε ως παράμετρο). Στη θέση αυτή της εντολής χρησιμοποιείται μια άλλη που πραγματοποιεί το λεγόμενο "indirect call". Αυτή η εντολή δέχεται σαν παράμετρο μια διεύθυνση, στην οποία βρίσκεται η διεύθυνση της συνάρτησης που θέλουμε να καλέσουμε. Δηλαδή, όπως δηλώνει και το όνομά της, η εν λόγω εντολή πραγματοποιεί έμμεσες κλήσεις! Παρεμπιπτόντως, ο γράφων συνηθίζει να ονομάζει αυτές τις κλήσεις και "patch calls". Όπως αντιλαμβάνεστε, κάθε φορά που καλούμε μια εξωτερική συνάρτηση (μέλος κάποιας βιβλιοθήκης) ο compiler χρησιμοποιεί ένα patch call και δίνει ως παράμετρο μια διεύθυνση εντός του import table. Σε αυτή τη διεύθυνση, όταν εκτελέσουμε το πρόγραμμα θα συμπληρωθεί αυτόματα η διεύθυνση μνήμης της ζητούμενης συνάρτησης.

Σε αυτό το σημείο οφείλουμε να κάνουμε μια διευκρίνιση: Η δομή του import table δεν είναι τόσο απλή όσο την περιγράψαμε προηγουμένως. Η "απλοποίηση" έγινε για

να εστιάσετε στη βασική λειτουργία και στη χρησιμότητα του πίνακα, χωρίς να πελαγώσετε με τις λεπτομέρειες της υλοποίησής του.



Εικόνα 6
Στην εικόνα βλέπουμε όλες τις βιβλιοθήκες που απαιτεί για τη λειτουργία του το "κομπιουτεράκι" των Windows (calc.exe). Όπως βλέπετε, δεν είναι και λίγες! Στην εικόνα έχουμε επιλέξει τη βιβλιοθήκη Kernel32.dll και το εργαλείο που χρησιμοποιούμε εμφανίζει όλες τις συναρτήσεις που περιλαμβάνονται σ' αυτήν.

Πρωτεύων στόχος

Χωρίς το import table είναι αδύνατο να εκτελεστεί μια εφαρμογή. Ακριβώς γι' αυτό, οι ισχυροί patchers φροντίζουν να παρεμβαίνουν πάντα σ' αυτό το κρίσιμο "εξάρτημα" των προγραμμάτων. Σε γενικές γραμμές, το ζητούμενο είναι η πλήρης απόκρυψη του import table και η δυναμική επανεμφάνιση *μόνο* των τμημάτων που χρειάζονται κάθε φορά. Με αυτόν τον τρόπο, όταν το προστατευμένο πρόγραμμα *δεν* εκτελείται, μοιάζει να μην έχει καθόλου import table! Έτσι, αν κάποιο εργαλείο ξεκλειδώματος απομακρύνει τον κώδικα του patcher (άρα και τις ρουτίνες που αναδημιουργούν το import table), το εκτελέσιμο δεν θα μπορεί να λειτουργήσει. Με λίγα λόγια, η επέμβαση των patchers στο import table έχει έναν απλό στόχο: Το να καταστεί αναγκαία για τη λειτουργία του προγράμματος η παρουσία του εμφυτευμένου κώδικα. Στη συνέχεια θα δούμε πώς εργάζεται ένας patcher για να πετύχει αυτό το στόχο.

Αρχικά, πραγματοποιείται μια ανάλυση του εκτελέσιμου κατά την οποία καταγράφονται όλες οι βιβλιοθήκες και όλες οι συναρτήσεις που αναφέρει το import table. Αυτά τα δεδομένα τοποθετούνται σ' έναν πίνακα διαφορετικής δομής (που ορίζει αυθαίρετα ο δημιουργός του patcher), σε κάποια "άσχετη" θέση εντός του εκτελέσιμου και, φυσικά, σε κρυπτογραφημένη μορφή. Στη συνέχεια τα πεδία του import table

μηδενίζονται, ώστε να μη μείνει κανένα ίχνος των αρχικών πληροφοριών. Τέλος, το παλιό import table καταργείται και στη θέση του δημιουργείται ένα νέο. Σε αυτό αναφέρονται μόνο τρεις συναρτήσεις, οι οποίες βρίσκονται στον κώδικα που εμφυτεύει ο patcher. Η μία από αυτές τις συναρτήσεις μπορεί να φορτώνει τις βιβλιοθήκες που έχουν καταγραφεί στον κρυμμένο πίνακα, ενώ η άλλη μπορεί να εντοπίζει τις διευθύνσεις των απαιτούμενων συναρτήσεων και να συμπληρώνει κατάλληλα το import table. Η τρίτη συνάρτηση εκτελείται κατά την εκκίνηση του προγράμματος και μόνο μία φορά, για να ελέγξει σε ποια διεύθυνση έχει φορτωθεί το ίδιο το εκτελέσιμο. Αυτή η διεύθυνση πρέπει να είναι γνωστή στον κώδικά του patcher, καθώς αποτελεί το σημείο αναφοράς για τα δεδομένα του προγράμματος, όπως επίσης και για τα relocation που ενδέχεται να πραγματοποιηθούν. Επιγραμματικά, η λογική που πρέπει να ακολουθεί ο εμφυτευμένος κώδικας για την ανασύνθεση του import table έχει ως εξής:

1. Αποκρυπτογράφηση ενός τμήματος του κρυμμένου πίνακα, από όπου θα διαβαστεί το όνομα μιας βιβλιοθήκης.
2. Φόρτωση της βιβλιοθήκης.
3. Διαγραφή της περιοχής της μνήμης, από όπου διαβάστηκε το όνομα της βιβλιοθήκης.
4. Αποκρυπτογράφηση ενός τμήματος του κρυμμένου πίνακα, από όπου θα διαβαστεί το όνομα μιας συνάρτησης (της φορτωμένης βιβλιοθήκης).
5. Εντοπισμός διεύθυνσης του κώδικα της συνάρτησης.
6. Μηδενισμός της περιοχής μνήμης, από όπου διαβάστηκε το όνομα της συνάρτησης.
7. Τοποθέτηση της διεύθυνσης του κώδικα της συνάρτησης στην κατάλληλη θέση του import table.
8. Αν χρησιμοποιείται και άλλη συνάρτηση από την ίδια βιβλιοθήκη, επίστρεψε στο βήμα 4.
9. Αν χρησιμοποιείται και άλλη βιβλιοθήκη, επίστρεψε στο βήμα 1.

Εδώ πρέπει να υπογραμμίσουμε μια λεπτομέρεια που δεν φαίνεται στη λίστα βημάτων: Ο δυναμικός σχηματισμός του import table πρέπει να πραγματοποιείται τμηματικά και οι περιοχές που δεν χρειάζονται κάθε φορά πρέπει να διαγράφονται αμέσως. Κάτι τέτοιο είναι απαραίτητο, ώστε να μην σχηματιστεί ολόκληρο το import table μέσα στη μνήμη. Αν συμβεί κάτι τέτοιο, ο επίδοξος cracker θα μπορέσει να αντιγράψει τον πίνακα και όλος ο κόπος του patcher θα πάει χαμένος!

Ένας patcher που υλοποιεί την παραπάνω διαδικασία σωστά, υλοποιεί κι ένα από τα σημαντικότερα συστήματα προστασίας. Ακόμη κι αν δεν περιλαμβάνει κανένα άλλο κλείδωμα, πρόκειται για έναν ισχυρότατο patcher ο οποίος προσφέρει ικανοποιητική προστασία στις εφαρμογές μας.

Παρεμβολές στις κλήσεις

Αν και θα μπορούσαμε να σταματήσουμε εδώ, πιστεύουμε ότι αξίζει να αναφέρουμε μερικά ακόμα τεχνάσματα των patchers. Όταν αναλύσαμε τη χρησιμότητα του import table, αναφερθήκαμε και στα patch calls: Κάθε φορά που καλούμε τη συ-

νάρτηση μιας βιβλιοθήκης μέσα από τον κώδικά μας, ο compiler χρησιμοποιεί μια εντολή έμμεσης κλήσης κι όχι άμεσης. Σε αυτή την εντολή, ο compiler δίνει σαν παράμετρο το πεδίο του import table που αντιστοιχεί στη ζητούμενη συνάρτηση. Από εκεί, όταν το πρόγραμμα εκτελείται και το import table έχει συμπληρωθεί, λαμβάνεται η διεύθυνση της συνάρτησης και πραγματοποιείται τελικά η κλήση. Όπως υποψιάζεστε, ένας patcher μπορεί να αλλοιώσει τις παραμέτρους των patch calls, με σκοπό την προσωρινή τους αχρήστευση. Φυσικά, ο κώδικας που εμφυτεύεται στο κλειδωμένο πρόγραμμα θα φροντίζει να διορθώσει τα "χαλασμένα" patch calls κι έτσι το πρόγραμμα θα λειτουργεί κανονικά. Αν κάποιος cracker ξηλώσει τον κώδικα που εμφύτευσε ο patcher, τα patch calls δεν θα διορθωθούν και οι κλήσεις προς τις διάφορες συναρτήσεις δεν θα μπορούν να πραγματοποιηθούν. Με λίγα λόγια, αν ξηλωθεί ο κώδικας του patcher το πρόγραμμα θα πάψει να λειτουργεί. Ας δούμε πώς υλοποιείται αυτό το τέχνασμα.

Ο patcher ανατρέχει σε όλα τα sections που είναι δηλωμένα σαν εκτελέσιμα (executable) κι άρα περιλαμβάνουν κώδικα. Αναζητά τα OpCodes που αντιστοιχούν στην εντολή έμμεσων κλήσεων (indirect call) και στην εντολή έμμεσων μεταβάσεων (indirect jump). Σε αυτές τις δύο εντολές, ο compiler δίνει πάντα ως παράμετρο μια διεύθυνση εντός του import table. Κάθε φορά που ο patcher εντοπίζει τα σχετικά OpCodes (0xFF15 για τις έμμεσες κλήσεις και 0xFF25 για τις έμμεσες μεταβάσεις), διαβάζει το DWORD που βρίσκεται ακριβώς δίπλα. Αν αυτό το DWORD *δεν* παραπέμπει κάπου μέσα στο import table, θεωρούμε ότι *δεν* έχουμε εντοπίσει τα ζητούμενα OpCodes. Συγκεκριμένα, θεωρούμε ότι έχουμε συναντήσει κάποια άσχετα bytes, που έτυχε να έχουν ίδια τιμή με κάποιο από τα OpCodes. Κάπου εδώ, όμως, τίθεται ένα εύλογο ερώτημα: Πώς θα ελέγξουμε αν το DWORD παραπέμπει εντός του import table; Επειδή αυτές οι διευθύνσεις είναι πάντοτε σχετικές (RVA), το εν λόγω νούμερο θα πρέπει να ανήκει στο διάστημα μεταξύ των τιμών ImportTable και (ImportTable + ImportSize). Το μέγεθος ImportTable αποτελεί τη σχετική διεύθυνση εκκίνησης του import table, ενώ το ImportSize δηλώνει το μέγεθος του import table. Σημειώστε ότι και τα δύο αυτά μεγέθη αναφέρονται στο PE32 header. Ας μην ξεφεύγουμε, όμως. Όταν ο patcher βεβαιωθεί ότι εντόπισε το OpCode ενός patch call, αποθηκεύει την διεύθυνση στην οποία βρέθηκε η εντολή, καθώς και την παράμετρο που είχε δώσει στην εντολή ο compiler. Επιπρόσθετα, η αρχική παράμετρος της εντολής αντικαθίσταται από μια τυχαία τιμή. Αυτή η διαδικασία επαναλαμβάνεται για όλα τα patch calls που εντοπίζονται. Έτσι, στο τέλος της διαδικασίας δημιουργείται ένας πίνακας με τις θέσεις *όλων* των patch calls και τις αντίστοιχες παραμέτρους. Ταυτόχρονα, οι σχετικές εντολές έχουν λάβει από μια τυχαία παράμετρο και υπό μία έννοια έχουν αχρηστευτεί. Περιττό να πούμε ότι ο πίνακας με τις θέσεις των εντολών και τις σωστές παραμέτρους αποθηκεύεται σε κρυπτογραφημένη μορφή! Κατά την εκτέλεση του προγράμματος, ο εμφυτευμένος κώδικας ανατρέχει στον πίνακα και διορθώνει τα patch calls on-the-fly.

Σημειώστε ότι αυτό το τέχνασμα είναι αρκετά ριψοκίνδυνο και σε ορισμένα εκτελέσιμα ενδέχεται να δημιουργήσει προβλήματα. Γι' αυτό οι patchers που το υλοποιούν, παρέχουν και τη δυνατότητα απενεργοποίησής του.

Πλήρης εξαφάνιση!

Είναι γνωστό ότι υπάρχουν εργαλεία που μπορούν να ανασυνθέσουν το import table, στα εκτελέσιμα που έχουν γίνει dump από τη μνήμη. Ακριβώς γι' αυτό, ένας patcher οφείλει να κρύβει όλα τα δεδομένα που σχετίζονται με το import table ή,

τέλος πάντων, όσο περισσότερα μπορεί. Κάτι τέτοιο μπορεί να εξασφαλιστεί με τα ακόλουθα βήματα:

- Καταγραφή των βιβλιοθηκών και των συναρτήσεων που έχουν δηλωθεί στο import table.
- Αποθήκευση των ονομάτων των βιβλιοθηκών και των αντίστοιχων συναρτήσεων, σε κάποια άσχετη θέση, με μια δομή της δικής μας επινόησης και σε κρυπτογραφημένη μορφή.
- Διαγραφή των περιεχομένων του import table.
- Δημιουργία ενός νέου import table σε νέα θέση, στο οποίο θα περιέχονται μόνο οι τρεις συναρτήσεις που αναφέραμε και νωρίτερα.
- Καταγραφή της θέσης και της παραμέτρου όλων των patch calls.
- Αποθήκευση του σχετικού πίνακα σε κρυπτογραφημένη μορφή.
- Διαγραφή της αρχικής (και άρα σωστής) παραμέτρου σε κάθε patch call.

Αν περιοριστούμε στα παραπάνω, το μόνο που θα πετύχουμε είναι η καταστροφή του εκτελέσιμου. Για τη σωστή λειτουργία του προστατευμένου προγράμματος, ο κώδικας που εμφυτεύεται από τον patcher θα πρέπει να εκτελεί τις "αντίστροφες" λειτουργίες:

- Δέσμευση μιας περιοχής μνήμης.
- Επιλογή μιας τυχαίας θέσης μέσα στην δεσμευμένη περιοχή, για την κατασκευή του νέου import table.
- Φόρτωση των απαραίτητων βιβλιοθηκών και εντοπισμός της διεύθυνσης των αντίστοιχων συναρτήσεων.
- Συμπλήρωση του νέου import table με τα σωστά δεδομένα.
- Επαναφορά των αρχικών παραμέτρων στα patch calls και διόρθωση αυτών των τιμών, ώστε να παραπέμπουν στο νέο import table.

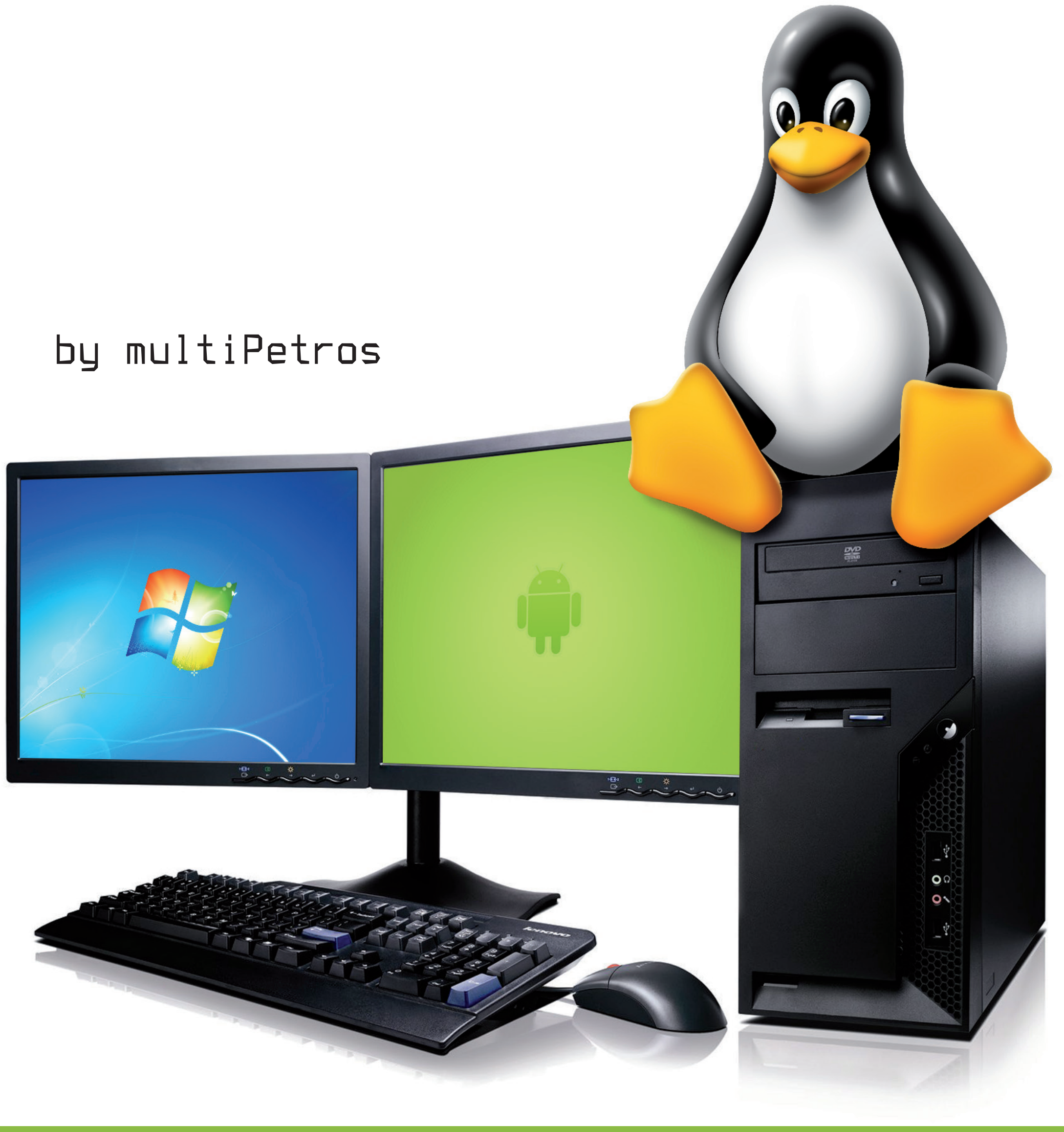
Κι όποιος αντέξει...

Τι θα λέγατε για ένα ακόμα εμπόδιο στην εργασία του cracker; Μετά από όσα έχουμε δει, θα μπορούσαμε να παρουσιάσουμε και μια τεχνική anti-debugging. Εντοπίζουμε όλα τα patch calls και μετράμε πόσες φορές καλείται κάθε συνάρτηση. Ακολούθως, επιλέγουμε εκείνη που χρησιμοποιείται λιγότερο από όλες. Αμέσως μετά βρίσκουμε όλα τα patch calls προς τη συγκεκριμένη συνάρτηση και τα αντικαθιστούμε με κάποια εντολή που γνωρίζουμε ότι θα προκαλέσει exception! Στο τέλος δημιουργούμε και έναν exception handler, ο οποίος καλεί τη συνάρτηση. Σατανικό, έτσι δεν είναι;

Ελπίζουμε η συναρπαστική μας πορεία στα ενδότερα των εκτελέσιμων PE32 και των patchers να σας γοήτευσε όσο γοητεύει κι εμάς. Μην νομίζετε όμως ότι έχουμε εξαντλήσει το θέμα! Δεν αποκλείεται να επανέλθουμε στο μέλλον, με περισσότερα κόλπα που θα προκαλέσουν πονοκέφαλο στους επίδοξους crackers.

Skill: Beginner
Tags: Android, x86, port, dual boot, GParted

by multiPetros



Ένα πράσινο ρομπότ για το PC σας [μέρος 1]

Με σύμμαχο ένα οικοσύστημα που απαρτίζεται από μια πληθώρα εφαρμογών για κάθε χρήση και γούστο, το Android έχει αδιαμφισβήτητα κατακλύσει την αγορά των φορητών συσκευών. Από κινητά τηλέφωνα και tablets, έως TV boxes και entertainment devices, το Android έχει ισχυρότατη παρουσία. Μήπως ήρθε η στιγμή να μπει κι ως επιλογή στο boot menu του υπολογιστή σας;

Για το Android, το λειτουργικό σύστημα της Google για φορητές συσκευές, δεν χρειάζονται ιδιαίτερες συστάσεις. Αποτελεί το δημοφιλέστερο λειτουργικό σύστημα κατέχοντας, σύμφωνα με αναλυτές της αγοράς, ποσοστό άνω του 80% επί του συνόλου των κυκλοφορούντων "έξυπνων" φορητών συσκευών. Πέρα από τα κινητά τηλέφωνα στα οποία πρωτοαπευθύνθηκε, από την αφανή έκδοση Honeycomb (3.0) κι έπειτα το Android ενσωμάτωσε χαρακτηριστικά με σαφή προσανατολισμό προς την αγορά των tablets. Από την έκδοση Ice Cream Sandwich και μετά, η αγορα το εναγκάλισε ενθέρμως. Η ενθουσιώδης στροφή του κοινού προς τα tablets από τη στιγμή που εμφανίστηκαν προτάσεις για κάθε βαλάντιο, γεγονός στο οποίο συνέβαλε και το ίδιο το Android, αφάνισε ένα σημαντικό κομμάτι της αγοράς των φορητών υπολογιστών. Συγκεκριμένα, η θεαματική μείωση των πωλήσεων στην κατηγορία των Netbooks και Subnotebooks, ώθησε τους κατασκευαστές στην απόφαση για διακοπή της παραγωγής των πρώτων και στον σημαντικό περιορισμό των δεύτερων.

Ένας φίλος απ' τα παλιά

Πιθανώς να το θυμόσαστε: Την εποχή που τα Netbook πουλούσαν σαν το ζεστό ψωμί, πολλές κατασκευάστριες εταιρείες μαζί με το κύριο λειτουργικό σύστημα που είχαν εγκατεστημένο στα μηχανήματα —και το οποίο στην πλειονότητα των περιπτώσεων ήταν τα Windows—, είχαν και μια δική τους, προσαρμοσμένη διανομή Linux. Αυτή διέθετε ένα ιδιαιτέρως απλοποιημένο interface, προσανατολισμένο στην αναπαραγωγή multimedia και στην πλοήγηση στο Διαδίκτυο. Η εισαγωγή στο δεύτερο αυτό λειτουργικό γινόταν είτε κατά την εκκίνηση μέσω του boot manager, είτε ενεργοποιώντας το Netbook από κάποιο ειδικό κουμπί. Σκοπός ύπαρξης της εκάστοτε custom Linux διανομής, δεν ήταν άλλος από την πιο γρήγορη εκκίνηση και κατά συνέπεια πιο γρήγορη χρήση του υπολογιστή, σε σχέση με την κλασική διαδικασία φόρτωσης του κύριου λειτουργικού συστήματος. Οι διανομές αυτές σπάνια ικανοποιούσαν τους χρήστες τους, καθώς οι μεν power users τις έβρισκαν ιδιαιτέρως περιοριστικές, ενώ οι λεγόμενοι "απλοί χρήστες" δεν χαίρονταν ιδιαίτερα στην ιδέα εκμάθησης ενός νέου τρόπου εργασίας. Τα καλούδια αυτά, λοιπόν, στην καλύτερη των περιπτώσεων έμεναν συνήθως να σκονίζονται στο partition που ήταν εγκατεστημένα. (Σ.τ.Ε. Από ψηφιακή σκόνη, φυσικά :P)

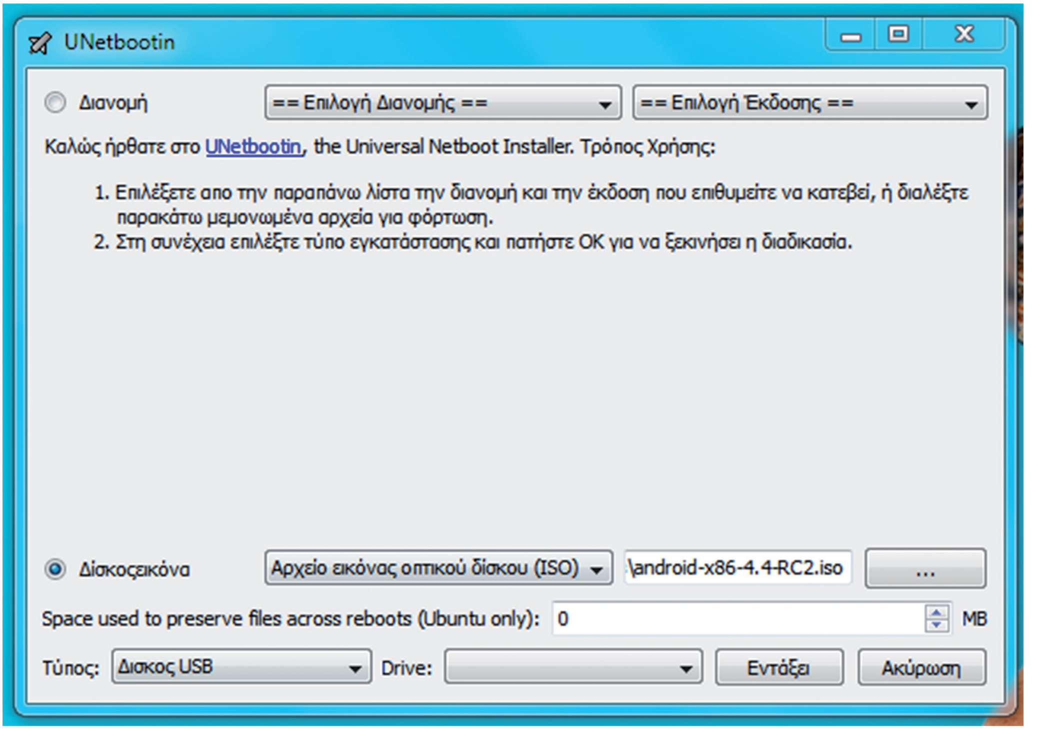
Επιστροφή στο μέλλον

Από τη στιγμή που η διαχωριστική γραμμή ανάμεσα στα tablets και τους φορητούς υπολογιστές όλο και ξεθωριάζει[1], δεν αποτέλεσαν έκπληξη οι απόπειρες κυκλοφορίας στην αγορά φορητών συσκευών που φέρουν τόσο Windows, όσο και Android. Φαεινό παράδειγμα είναι η σειρά Transformer της ASUS. Από τη μια όμως το υψηλό κόστος αυτών των προτάσεων, από την άλλη κυρίως η αποστροφή των γιγάντων της πληροφορικής (βλέπε Google και Microsoft) προς μια τέτοια προοπτική, φανερώνει ότι προς το παρόν οι προτάσεις αυτές δεν θα πολλαπλασιαστούν γρήγορα. Όλα όμως αυτά έχουν να κάνουν με πολιτικές και στρατηγικές εταιρειών κι ελάχιστα απασχολούν τους χρήστες με όρεξη και μεράκι. Με σύμμαχο την απόφαση της Intel να μπει στην αγορά των φορητών συσκευών με τα BayTrail SoCs, το Android ήρθε ακόμα πιο κοντά στα PCs και συγκεκριμένα στην πλατφόρμα x86.

Με δεδομένη λοιπόν αυτή την κατάσταση, και βλέποντας ότι το Android είναι ό,τι πρέπει για να πάρει τη θέση ενός δευτερεύοντος multimedia και Internet OS, με

1. Touch χαρακτηριστικά των Windows 8, εμφάνιση οικονομικών laptops με οθόνες αφής, tablets με ολοένα και περισσότερες επεξεργαστικές δυνατότητες κ.λπ.

ιδιαιτέρως μεγάλη βάση χρηστών κι εφαρμογών, η κοινότητα των προγραμματιστών επιφορτίστηκε με την ανάπτυξη μιας διανομής του Android για προσωπικούς υπολογιστές. Το αποτέλεσμα είναι το project που ακούει στο όνομα Android x86. Το εν λόγω εγχείρημα αποτελεί τη μετεξέλιξη της συλλογής διάφορων patches που βοηθούσαν στο porting του Android στην πλατφόρμα x86. Στο site του project, στη διεύθυνση www.android-x86.org, υπάρχουν αρκετές πληροφορίες, links προς τον πηγαίο κώδικα και φυσικά προς τα έτοιμα για κάψιμο[2] ISO images. Μέχρι και η έκδοση ICS κυκλοφορούσε σε πολλά ISO images, για διάφορους τύπους δημοφιλών κυρίως netbooks. Από την έκδοση KitKat όμως κι έπειτα οι προγραμματιστές του project δημιούργησαν ένα universal ISO που καλύπτει πλήθος προτάσεων από την πλατφόρμα x86.



Το λιτό περιβάλλον του UNetbootin μας υποδέχεται, δίνοντάς μας μάλιστα τη δυνατότητα να κατεβάσει αυτό για λογαριασμό μας την επιθυμητή διανομή που θέλουμε να περάσουμε στο flash drive.

Δημιουργία bootable media με το Android x86

Αξίζει να αναφέρουμε ότι τα ISO images του Android x86 είναι live διανομές, οι οποίες πέρα από την άμεση φόρτωση κι εκτέλεση του λειτουργικού στη μνήμη του υπολογιστή, παρέχουν και τη δυνατότητα εγκατάστασής του σε κάποιο διαμέρισμα

2. Η διαδικασία εγγραφής αρχείων σε οπτικά μέσα πολλές φορές ονομάζεται "κάψιμο", διότι στην πραγματικότητα περί αυτού πρόκειται: Η ακτίνα του laser της συσκευής καίει την επιφάνεια του οπτικού δίσκου, χαράσσοντας σημεία αυτής ή αφήνοντάς την ανέπαφη, δημιουργώντας έτσι λακκούβες και νησίδες. Η αντανάκλαση του φωτός στην επιφάνεια του μέσου αντιστοιχεί στις καταστάσεις "μηδέν" και "ένα", δηλαδή σε δυαδικά ψηφία.

του σκληρού του δίσκου. Για τους σκοπούς του άρθρου θα χρησιμοποιήσουμε την τελευταία έκδοση, την 4.4-RC2. Κατεβάζουμε λοιπόν το σχετικό αρχείο του ISO image από τη διεύθυνση www.android-x86.org/download, το οποίο μάλιστα έχει σχετικά μικρό μέγεθος (για λειτουργικό σύστημα): περί τα 330MB. Να σημειώσουμε ότι το σύστημα στο οποίο κάναμε την εγκατάσταση είχε έναν δίσκο, με ένα διαμέρισμα, στο οποίο βρίσκονταν εγκατεστημένα τα Windows 8.

Επόμενο βήμα είναι η εγγραφή του ISO σε κάποιο μέσο που θα χρησιμοποιήσουμε κατά την εκκίνηση του υπολογιστή. Υπάρχουν δύο βασικές επιλογές: το κάψιμο του ISO image σε κάποιο οπτικό μέσο όπως CD ή DVD, ή η εγγραφή του σε κάποιο flash drive. Η πρώτη επιλογή δεν έχει κάποια ιδιαιτερότητα και μπορούμε να την υλοποιήσουμε απλά χρησιμοποιώντας το αγαπημένο μας πρόγραμμα εγγραφής CD/DVD. Η επιλογή όμως αυτή δεν είναι κατάλληλη για όλες τις περιπτώσεις, καθώς από τη μια τα περισσότερα netbooks και subnotebooks δεν έχουν CD/DVD drive, ενώ από την άλλη οι οπτικοί δίσκοι δεν έχουν την ευελιξία και τη μεταφερσιμότητα ενός flash drive.

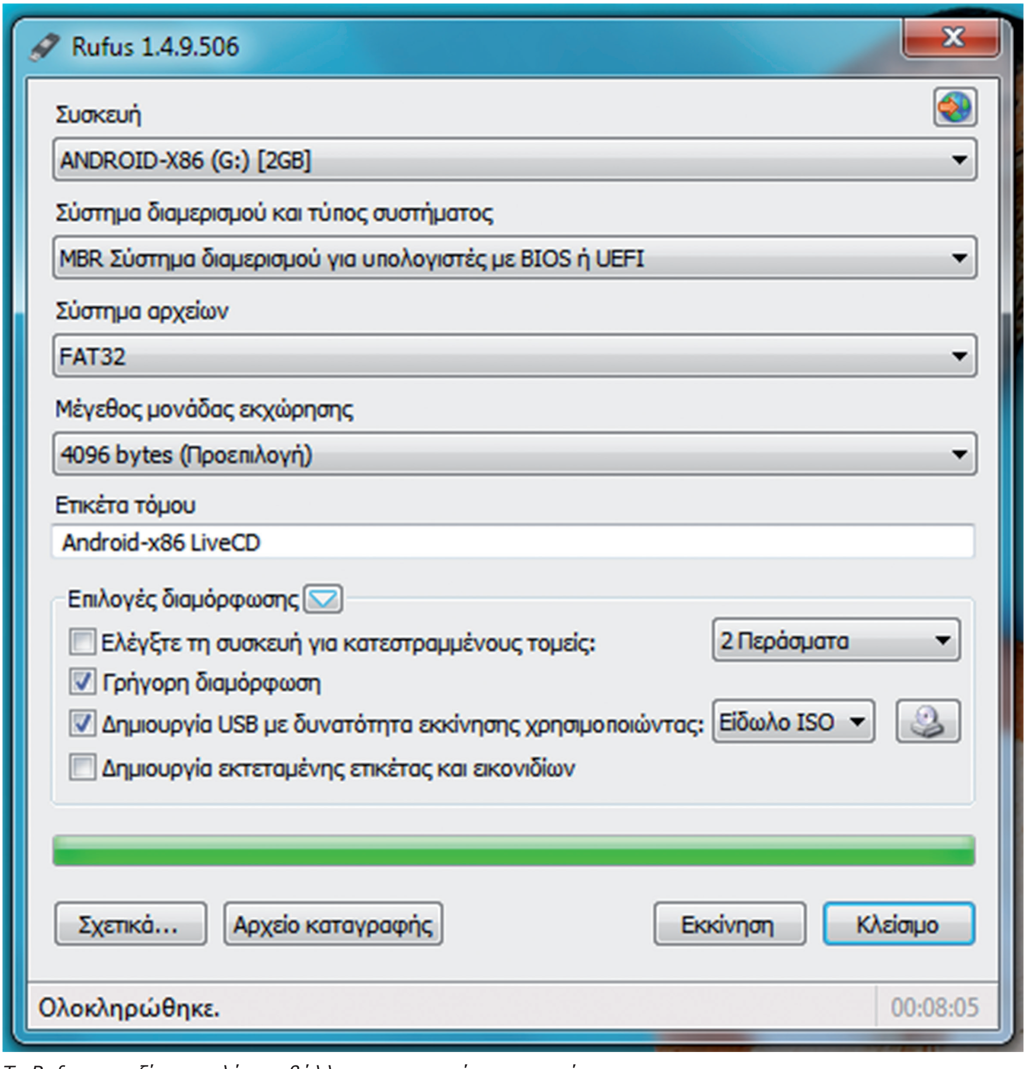
Έτσι, και μιας και με τις περιπέτειες έχουμε μια σχέση λατρείας, αποφασίσαμε να επιλέξουμε τη δημιουργία ενός bootable flash drive με το ISO image του Android x86. Οι διαθέσιμες επιλογές αυτής της περίπτωσης είναι αρκετές, αλλά αυτό κάθε άλλο παρά ανησυχία μας δημιουργεί.

Μπορούμε, λοιπόν, να γράψουμε το ISO image στο flash drive μας, με τη βοήθεια του UNetbootin (Universal Netboot Installer), το οποίο αποτελεί ένα εργαλείο εύκολης δημιουργίας live flash drives με διάφορες διανομές του Linux. Από το περιβάλλον του προγράμματος, το οποίο παρεμπιπτόντως μπορούμε να κατεβάσουμε από τη διεύθυνση unetbootin.sourceforge.net, έχουμε τη δυνατότητα επιλογής της επιθυμητής διανομής προς εγκατάσταση, το ISO image της οποίας μάλιστα αναλαμβάνει να κατεβάσει το ίδιο το πρόγραμμα. Βέβαια, υπάρχει και η δυνατότητα επιλογής του σχετικού αρχείου ISO από τον σκληρό μας δίσκο. Κι αυτό ακριβώς κάνουμε, καθώς η διανομή του Android x86 δε βρίσκεται στη σχετική λίστα της εφαρμογής. Κατόπιν, πατώντας στο "Εντάξει" το πρόγραμμα αναλαμβάνει τη δημιουργία ενός εκκινήσιμου flash drive με το Android x86, φέρνοντας όμως το boot menu στα δικά του μέτρα — περισσότερο ως προς την όψη, παρά ως προς την ουσία και τις λειτουργίες.

Μια άλλη προσέγγιση, η οποία μας παρέχει εμπειρία εγγύτερη σ' αυτή του καψίματος του ISO image σ' ένα CD ή DVD, είναι η χρήση του εργαλείου Rufus, το οποίο εγγράφει το ISO image στο flash drive δίχως όμως να κάνει παρεμβάσεις στον bootloader. Οι προγραμματιστές του Rufus, επίσης, διατείνονται ότι το δημιούργημά τους είναι γρηγορότερο σε σχέση με UNetbootin αλλά και σε σχέση με το RMPrepUSB, που είχαμε δει παλαιότερα[3]. Επιπλέον το Rufus, πέρα από τα ISO images διανομών Linux υποστηρίζει κι αυτά των Windows. Τέλος, μπορεί να συνεργαστεί άψογα και με images που απαιτούν ή υποστηρίζουν UEFI[4] αντί του κλασικού BIOS. Εμείς, πάντως, προτιμήσαμε το Rufus για τη δημιουργία του bootable flash drive μας, κυρίως επειδή θέλαμε να έχουμε μια περισσότερο αυθεντική εμπειρία γύρω από το Android x86 — ακόμα και στο επίπεδο του boot manager.

Το περιβάλλον του Rufus είναι εξίσου απλό με αυτό του UNetbootin κι αποτελείται από ένα παράθυρο, από το οποίο μπορούμε να επιλέξουμε το μέσο στο οποίο θέλουμε να εγγράψουμε το ISO image, το ίδιο το image φυσικά, καθώς και τις

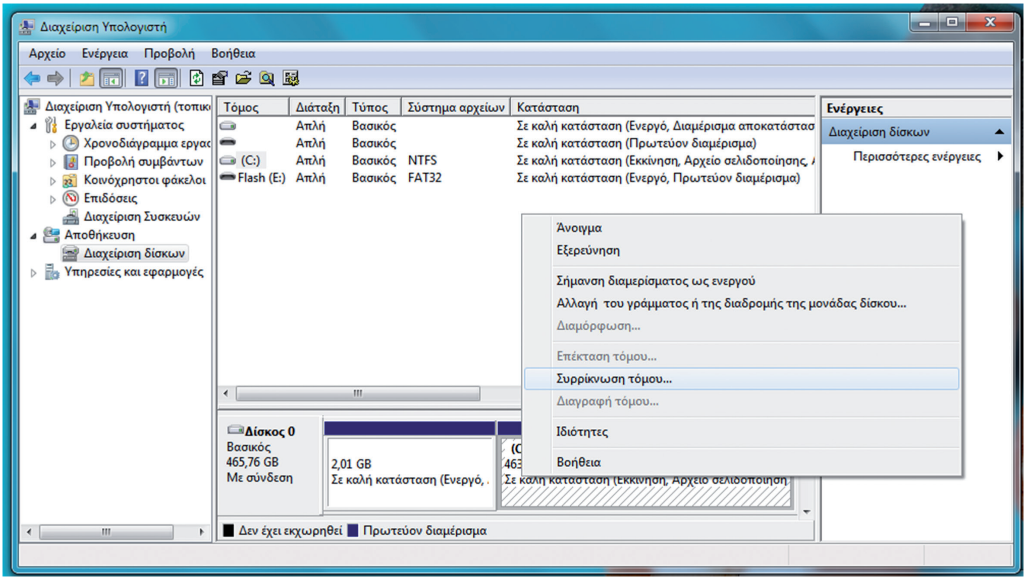
3. Τεύχος deltaHacker 021, άρθρο "Για μια χούφτα διανομές"
4. Unified Extensible Firmware Interface: Το νέο πρότυπο για την αντικατάσταση του κλασικού BIOS.



Το Rufus, με εξίσου απλό περιβάλλον, μας επιτρέπει να περάσουμε στο flash drive μας το ISO image που έχουμε κατεβάσει, ταχύτατα και δίχως παρεμβάσεις στο μενού εκκίνησής του.

ρυθμίσεις διαμόρφωσης του flash drive. Για τις ανάγκες των δοκιμών μας επιλέξαμε MBR partitioning για συστήματα με BIOS (αλλά και UEFI) κι ως σύστημα αρχείων το FAT32. Κατόπιν, αναζητήσαμε το ISO image στη θέση που το είχαμε αποθηκεύσει, πατώντας το σχετικό κουμπί με το εικονίδιο του δίσκου, αφού πρώτα είχαμε τικάρει την επιλογή "Δημιουργία USB με δυνατότητα εκκίνησης" κι ενεργοποιήσει την επιλογή "Είδωλο ISO", από το πλαίσιο επιλογής που υπάρχει δίπλα του. Τέλος, πατήσαμε το κουμπί "Εκκίνηση" και σε λίγο χρόνο είχαμε έτοιμο το bootable φλασάκι μας με το Android x86.

Κάπου εδώ, να επισημάνουμε ότι τόσο το UNetbootin όσο και το Rufus αποτελούν



Μέσα από την Διαχείριση Δίσκων των MS Windows, μπορούμε με ασφάλεια (για τη λειτουργία των ίδιων των Windows) να δημιουργήσουμε ένα διαμέρισμα που θα φιλοξενήσει το Android x86.

Λογισμικό Ανοιχτού Κώδικα, κι αυτό τους κερδίζει άλλον έναν πόντο στην καρδιά του γράφοντα.

Τέλος, για τους φίλους των UNIX συστημάτων, αφήσαμε τη χρήση του εργαλείου dd. Με το dd μπορούμε πανεύκολα και ταχύτατα να δημιουργήσουμε το bootable flash disk μας, δίνοντας σε μια κονσόλα κάτι σαν...

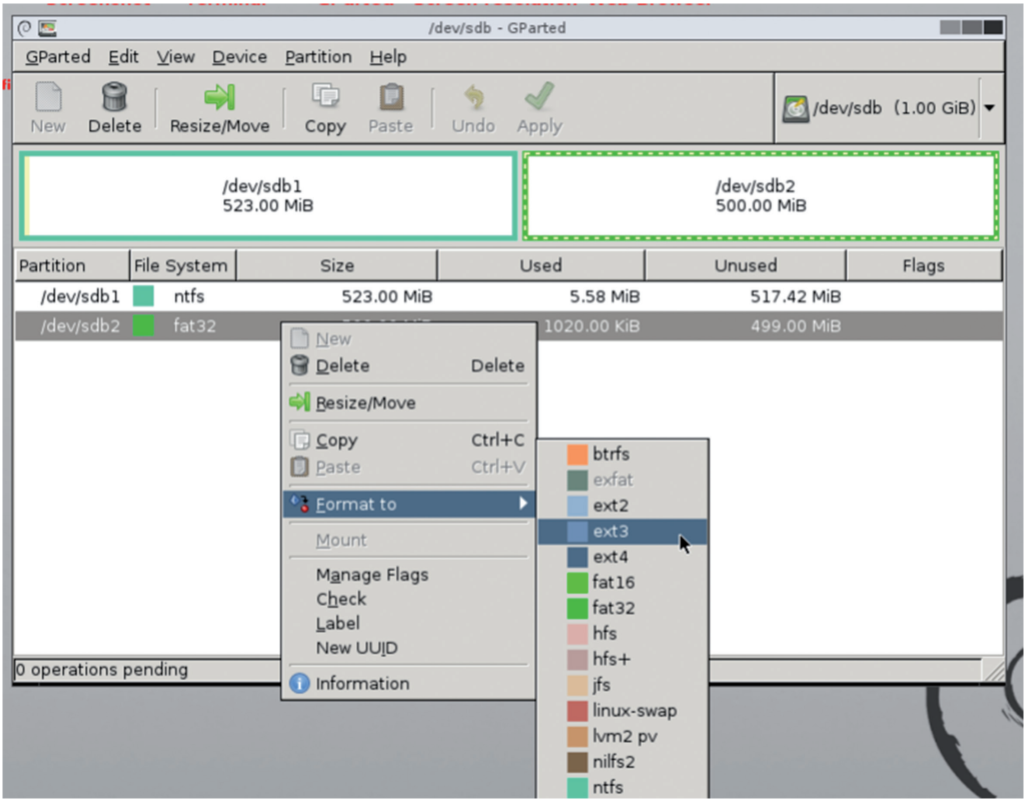
```
dd if=/path_to/android-x86-4.4-RC2.iso of=/dev/sdX
```

όπου /path_to/ είναι η διαδρομή προς το ISO image του Android x86 και sdX είναι το όνομα της USB συσκευής που αντιστοιχίστηκε στο flash drive μας, κατά τη σύνδεσή του στον υπολογιστή. Στον δικό μας, το όνομα της συσκευής ήταν /dev/sda1. Αυτό εύκολα το διαπιστώσαμε ελέγχοντας την έξοδο του dmesg, αφού συνδέσαμε το flash drive, δίνοντας:

```
dmesg | tail
```

Είμαστε καθ' όλα έτοιμοι λοιπόν να εκκινήσουμε το σύστημά μας από τη live διανομή του Android x86 που μόλις περάσαμε στο flash drive και, γιατί όχι, να την εγκαταστήσουμε σε κάποιο διαμέρισμα του σκληρού μας δίσκου. Ωπ, μια στιγμή! Γι' αυτό το τελευταίο σίγουρα δεν είμαστε έτοιμοι...

Θα πρέπει πρώτα να δημιουργήσουμε το σχετικό διαμέρισμα αλλά και να το διαμορφώσουμε, καθώς αν και τα σχετικά εργαλεία υπάρχουν *και* στη live distro του Android x86 δεν λειτουργούν όπως θα θέλαμε. Για τη δημιουργία του διαμερίσματος θα μπορούμε να χρησιμοποιήσουμε το GParted, μια διανομή Linux η οποία είναι εξειδικευμένη στο partitioning. Σε μια προσπάθεια όμως να αποφύγουμε προβλήματα που ενδέχεται να προκύψουν με την εγκατάσταση των Windows, θα καταφύγουμε στην Κονσόλα Διαχείρισης των Windows. Αμέσως μετά, για τη διαμόρφωση του δι-



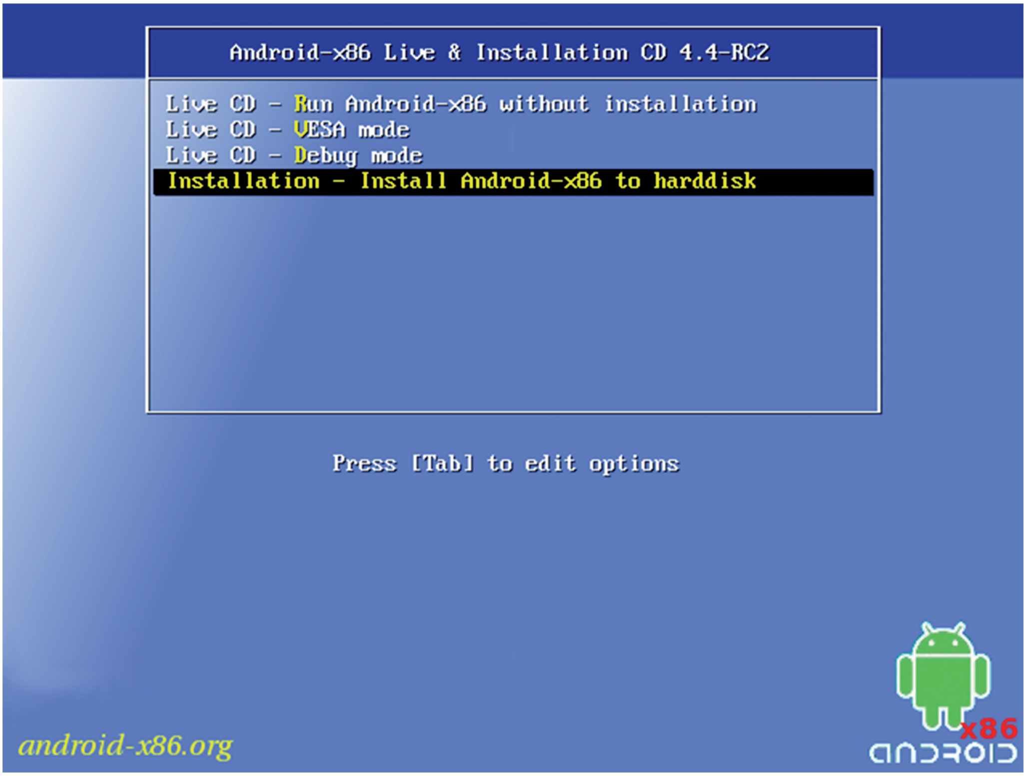
Η Live διανομή GParted με το ομώνυμο εργαλείο, μέσα από ένα φιλικότατο περιβάλλον μας επιτρέπει να μετατρέψουμε το σύστημα αρχείων του διαμερίσματός μας σε Ext3. Και, γιατί όχι, αν τολμάμε να τα βάλουμε με την επιδιόρθωση εκκίνησης των Windows, μπορούμε να δημιουργήσουμε το διαμέρισμα μέσω του ίδιου του GParted.

αμερίσματος που θα δεχτεί το Android θα χρησιμοποιήσουμε πράγματι το GParted, το οποίο μπορούμε να κατεβάσουμε στον υπολογιστή μας από τη διεύθυνση `http://gparted.org/download.php`.

Αφού ολοκληρωθεί το download (το μέγεθος της διανομής GParted είναι κι αυτό αρκετά μικρό, περί τα 120MB), θα πρέπει να περάσετε το ISO image σε ένα φλασάκι, με κάποιον απ' τους τρόπους που μόλις περιγράψαμε. Φυσικά, αν συγκαταλέγεστε στους φανατικούς αναγνώστες του περιοδικού, μπορεί να έχετε ήδη έτοιμο ένα bootable flash drive με τη βοήθεια του grub4dos (θα περιλαμβάνει *και* το GParted) όπως σας είχαμε δείξει στο σχετικό άρθρο του τεύχους 021.

Να επισημάνουμε εδώ ότι η διαμόρφωση του συστήματος αρχείων του διαμερίσματος μπορεί να γίνει με οποιοδήποτε σχετικό εργαλείο. Οι λόγοι που για τις ανάγκες του άρθρου επιλέξαμε το GParted αφορούν στην άδεια χρήσης του (Ελεύθερο Λογισμικό), στη δημοτικότητά του, στην ευχρηστία του και την υποστήριξη διαμόρφωσης κατά Ext3[5].

5. Περιγραφικά, ονομάζεται "third extended filesystem" και είναι από τα πιο συχνά χρησιμοποιούμενα συστήματα αρχείων στον κόσμο του Linux. Είναι αντικαταστάτης του παλαιότερου ext2 και, θεωρητικά, υποστηρίζει τόμους μεγέθους έως και 32TerraBytes. Τα τελευταία χρόνια δίνει τη θέση του στο νεότερο Ext4, με βελτιωμένα χαρακτηριστικά και επιδόσεις και υποστήριξη τόμων ενός ExaByte (~1.000.000 GigaBytes).



Το μενού εκκίνησης του ISO image της διανομής Android x86 παρέχει δυνατότητα για live δοκιμή ή για εγκατάσταση. Όπως κάνουν και οι περισσότερες Live διανομές Linux, δηλαδή.

Φυσικά, αν εσείς έχετε κάποιο διαφορετικό εργαλείο που προτιμάτε, μπορείτε ελεύθερα να το χρησιμοποιήσετε. Για παράδειγμα οι απανταχού φίλοι του Linux πιθανώς να προτιμήσουν τη χρήση του απλοϊκού mkfs.ext3, για τη γραμμή εντολών.

Προετοιμασία του δίσκου

Όσο δημιουργούνται τα bootable flash drives μας, ανοίγουμε τη Διαχείριση Δίσκων της Κονσόλας Διαχείρισης των Windows, γράφοντας στο πλαίσιο αναζήτησης του μενού "'Εναρξη" diskmgmt.msc. Στο παράθυρο που μας ανοίγει επιλέγουμε το δίσκο στον οποίο θέλουμε να δημιουργήσουμε το διαμέρισμα που θα φιλοξενήσει το Android x86. Αν το συνολικό χώρο του δίσκου καταλαμβάνει μια κατάτμηση, όπως συνέβη στη δική μας περίπτωση, πατάμε δεξί κλικ κι επιλέγουμε "Συρρίκνωση τόμου". Το μέγεθος του διαμερίσματος για το Android x86 δε χρειάζεται να είναι μεγάλο: Περί τα 20GB είναι κάτι παραπάνω από αρκετά. Για σύστημα αρχείων στο νέο διαμέρισμα μπορούμε, προσωρινά, να ορίσουμε το FAT32.

Μετά τη δημιουργία του διαμερίσματός μας κάνουμε reboot αλλά εκκινούμε το σύστημα από το flash drive που περιέχει το GParted. Αυτό το τελευταίο μπορούμε πολύ εύκολα να το πετύχουμε, πατώντας το κατάλληλο πλήκτρο τη στιγμή που ξεκινάει τη λειτουργία του ο υπολογιστής κι εμφανίζει τις πληροφορίες του BIOS. Συνήθως,

είναι κάποιο από τα [F9] έως [F12]. Αν πάντως το σύστημά μας δεν παρέχει σχετικό μενού, μπορούμε να εισέλθουμε στο περιβάλλον ρύθμισης του BIOS πατώντας το σχετικό πλήκτρο (συνήθως [DEL] ή [F1] ή [F2]) και στη σειρά εκκίνησης από συσκευές να βάλουμε το USB Drive μια θέση ψηλότερα από το σκληρό δίσκο.

Να επισημάνουμε ότι στην περίπτωση που δεν έχουμε Windows αλλά τρέχουμε κάποια διανομή Linux ή στην περίπτωση που είμαστε λίγο ριψοκίνδυνοι και γνωρίζουμε τη λειτουργία επιδιόρθωσης εκκίνησης των Windows, μπορούμε να δημιουργήσουμε το διαμέρισμα και μέσα από το ίδιο το περιβάλλον της διανομής GParted. Αρχικά, κάνουμε κλικ στο ομώνυμο εργαλείο–εικονίδιο που βρίσκεται σε περίοπτη θέση στην επιφάνεια εργασίας και μετά πατάμε στο κουμπί Resize/Move, που βρίσκεται στη μπάρα εργαλείων του προγράματος. Έτσι, μειώνουμε το μέγεθος της υπάρχουσας κατάτμησης και δημιουργούμε χώρο γι' αυτήν που θα φιλοξενήσει το Android x86. Στη συνέχεια, επιλέγοντας τον κενό χώρο και πατώντας στο κουμπί "New" μπορούμε να δημιουργήσουμε ένα νέο διαμέρισμα, έχοντας τσεκάρει το σύστημα αρχείων Ext3.

Εμείς διαλέξαμε πάντως την ασφαλή οδό, επομένως θα πρέπει τώρα να δημιουργήσουμε το σύστημα αρχείων του διαμερίσματος του Android x86 διαμορφώνοντας ξανά το διαμέρισμα που είχαμε φτιάξει με τη "Διαχείριση Δίσκων" των Windows. Απλά, αυτή τη φορά αντί για FAT32 θα επιλέξουμε το Ext3. Από το παράθυρο του προγράμματος GParted επιλέγουμε προσεκτικά το διαμέρισμα που θέλουμε να διαμορφώσουμε και με δεξί κλικ σ' αυτό πηγαίνουμε στο "Format to" κι επιλέγουμε τον τύπο Ext3. Επιβεβαιώνουμε την αλλαγή κι εκτελούμε τη μετατροπή με ένα πάτημα του πλήκτρου "Apply", πάνω στη γραμμή εργαλείων του προγράμματος, κι απαντώντας θετικά στη σχετική ερώτηση που θα μας τεθεί.

Κάπου εδώ, η προετοιμασία του δίσκου για τη φιλοξενία του Android x86 έχει ολοκληρωθεί και είμαστε πλέον έτοιμοι να προχωρήσουμε στην εγκατάστασή του!

Εγκατάσταση του Android x86

Τερματίζοντας τη λειτουργία του GParted (από το σχετικό εικονίδιο με τίτλο "Exit") ξεκινάμε το σύστημά μας — από το bootable flash drive του Android x86 αυτή τη φορά. Το μενού εκκίνησής του περιλαμβάνει τέσσερεις επιλογές. Η πρώτη αφορά στην εκτέλεση σε μορφή Live Distro, ώστε να πάρουμε μια γεύση του συστήματος δίχως να κάνουμε αλλαγές στον δίσκο, ενώ η τελευταία, στην εγκατάσταση της διανομής στο δίσκο μας. Εμείς, έχοντας σκοπό να χρησιμοποιούμε το Android x86 και να δημιουργήσουμε ένα dual boot σύστημα, διαλέξαμε την τελευταία επιλογή, αυτή της εγκατάστασής του.

Στη συνέχεια, κι αφού φορτωθεί το περιβάλλον εγκατάστασης της διανομής, το οποίο προς χαρά των φίλων του Unix θυμίζει έντονα τον FreeBSD installer, μας ζητείται να επιλέξουμε το διαμέρισμα του δίσκου όπου θα εγκατασταθεί το νέο μας λειτουργικό. Πολύ προσεκτικά, για μια ακόμη φορά επιλέγουμε το διαμέρισμα που μόλις πριν δημιουργήσαμε.

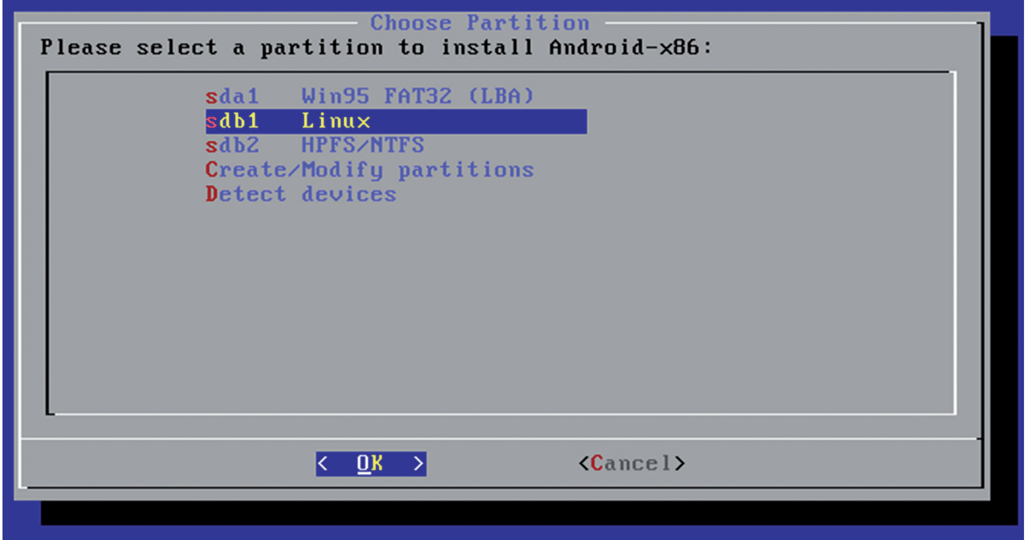
Στη συνέχεια ερωτόμαστε για το αν επιθυμούμε να γίνει διαμόρφωση του συστήματος αρχείων στο επιλεχθέν διαμέρισμα. Πολύ ευγενικά απαντούμε ότι δεν θέλουμε κάτι τέτοιο. Από τη μια το έχουμε ήδη διαμορφώσει, από την άλλη, σύμφωνα με τους προγραμματιστές του Android x86, η διαμόρφωση σε Ext3 από το περιβάλλον εγκατάστασης διανομής δεν λειτουργεί σωστά.

Στη συνέχεια θα ερωτηθούμε για την εγκατάσταση του boot manager ονόματι GRUB. Μιας και θέλουμε να έχουμε τον πλήρη έλεγχο στο σύστημά μας, αλλά και να αποφύγουμε τις όποιες περιπέτειες, παρόλο που οι δημιουργοί του Android x86 προσπαθούν να μας καθησυχάσουν περί του αντιθέτου επιλέγουμε "Skip". Στη συνέχεια, θα αναλάβουμε εμείς να ρυθμίσουμε τον υπάρχοντα boot loader του συστήματός μας για την παρουσία του Android x86.

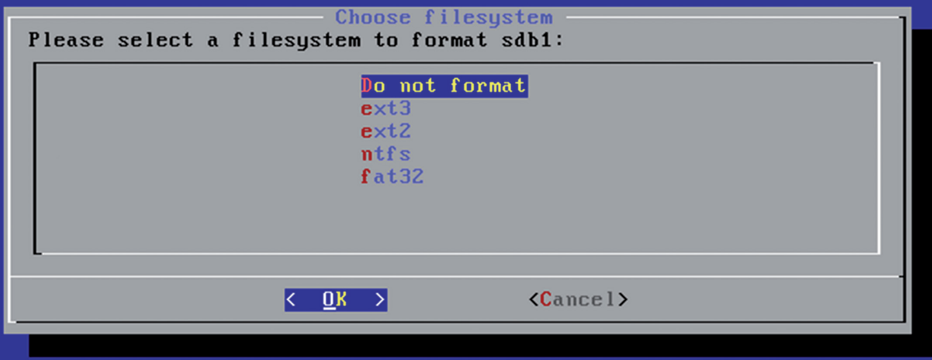
Άλλη μια ερώτηση εμφανίζεται στην οθόνη μας κι αυτή τη φορά έχει να κάνει με τη δυνατότητα εγγραφής στον φάκελο system. Αν θέλουμε να πειραματιστούμε αρκετά με το σύστημα, μπορούμε να απαντήσουμε θετικά. Το μόνο που θα χρειαστούμε είναι 300 περίπου επιπλέον MB στο δίσκο και λίγο παραπάνω χρόνο στην εγκατάσταση. Βλέπετε, αυτό που ουσιαστικά γίνεται είναι η εξαγωγή των αρχείων του εικονικού φακέλου system από την αρχειοθήκη που περιέχονται, σ' έναν πραγματικό φάκελο στο δίσκο μας.

Ύστερα κι απ' αυτή την ερώτηση ξεκινάει η εγκατάσταση των αρχείων στο δίσκο μας. Μετά την ολοκλήρωσή της, επιλέγουμε να επανεκκινήσουμε τον υπολογιστή μας ώστε να ολοκληρώσουμε τη δημιουργία του dual boot συστήματος, ρυθμίζοντας τον boot manager.

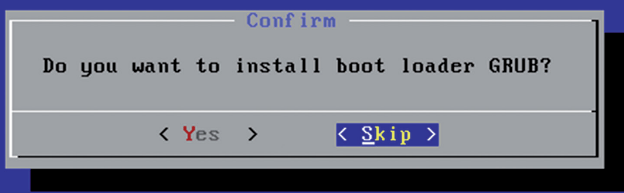
Ώρα όμως και για μια δική μας επανεκκίνηση. Μετά από ένα απαραίτητο, για την εποχή, χτύπημα φραπέ, μπορούμε να μεταβούμε στη σελίδα 56 του τεύχους, όπου βρίσκεται η συνέχεια του πρώτου μέρους του μίνι αφιερώματός μας.



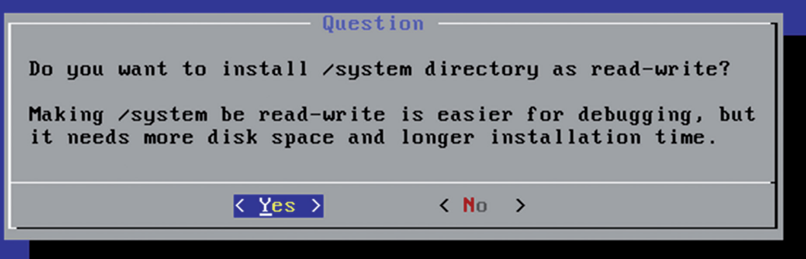
Το περιβάλλον εγκατάστασης του Android x86 μάς φέρνει στο νου κάτι από την εγκατάσταση του FreeBSD, δεν συμφωνείτε; Μην παρασυρθείτε ωστόσο από τη γοητεία του. Θα πρέπει να είμαστε ιδιαίτερα προσεκτικοί κατά στην επιλογή του διαμερίσματος του δίσκου, όπου θέλουμε να εγκαταστήσουμε το Android x86.



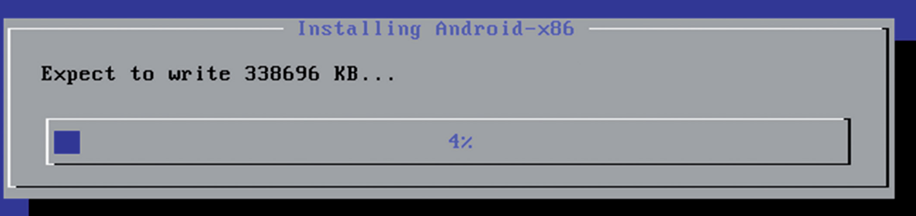
Από το περιβάλλον εγκατάστασης ερωτόμαστε αν επιθυμούμε να διαμορφωθεί για εμάς το σύστημα αρχείων, στο επιλεγμένο διαμέρισμα. Μιας κι έχουμε φροντίσει ήδη γι' αυτό, αρνούμαστε ευγενικά.



Το περιβάλλον εγκατάστασης μπορεί να διευθετήσει αυτόματα όλα τα σχετικά με τον boot loader ζητήματα. Συγνώμη, αλλά ούτε κι αυτή τη φορά θα πάρουμε :)



Η απάντηση στο ερώτημα δημιουργίας φακέλου system με δυνατότητα εγγραφής, επαφίεται στην ευγένεια αλλά κυρίως στη διάθεση για εμβάθυνση που έχουμε στο Android σύστημά μας.



Αφού έχουμε ολοκληρώσει τις ερωταπαντήσεις, ξεκινά η διαδικασία εγκατάστασης του συστήματος. Δεν πρόκειται να μας απασχολήσει για μεγάλο χρονικό διάστημα.

Skill: Advanced
Tags: code encryption, polymorphic code, crypto algorithm, crypto algorithm generators



by Jef

Γεννήτρια πολυμορφικού κώδικα κρυπτογράφησης

Στο τεύχος 032 αναφερθήκαμε στις εργασίες που πραγματοποιεί ένας patcher, στην προσπάθειά του να δυσκολέψει και τελικά να αποτρέψει το έργο ενός cracker. Σε αυτό το άρθρο πιστεύουμε ότι θα ήταν σκόπιμο να εμβαθύνουμε στο κομμάτι της κρυπτογράφησης του κώδικα. Η κρυπτογράφηση βέβαια είναι έννοια γνωστή, αλλά μέχρι τώρα εξετάζαμε μόνο τις εφαρμογές της στα συστήματα επικοινωνίας και στην προστασία αρχείων ή και ολόκληρων δίσκων.

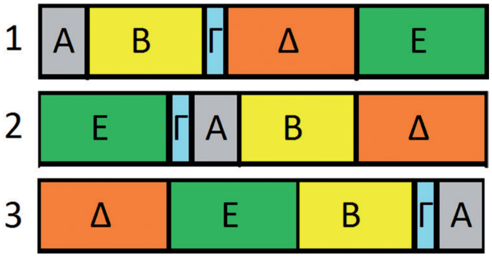
Η κρυπτογράφηση του κώδικα παρουσιάζει μια σημαντική ιδιαιτερότητα: Για να είναι δυνατή η εκτέλεση του κώδικα, το κρυπτογραφημένο εκτελέσιμο πρέπει να περιέχει και το κλειδί της αποκρυπτογράφησης! Αυτό σημαίνει ότι δεν έχουμε να κερδίσουμε τίποτα χρησιμοποιώντας κάποιον ισχυρό αλγόριθμο ασύμμετρης κρυπτογράφησης. Αν, για παράδειγμα, χρησιμοποιήσουμε τον αλγόριθμο RSA, ο επίδοξος cracker θα πάρει το κλειδί αποκρυπτογράφησης από το εκτελέσιμο και με τη βοήθεια κάποιας έτοιμης βιβλιοθήκης θα αποκρυπτογραφήσει τον κώδικα άνετα. Αυτό που μας συμφέρει, λοιπόν, είναι να χρησιμοποιήσουμε έναν δικό μας αλγόριθμο κρυπτογράφησης, ο οποίος θα αναγκάσει τον επιτιθέμενο να αναλύσει και να κατανοήσει τον κώδικά μας *πριν* καταφέρει να προχωρήσει στην αποκρυπτογράφηση. Επιπρόσθετα, όπως αναφέραμε και στο σχετικό άρθρο του προηγούμενου τεύχους, το κλειδί κρυπτογράφησης δεν πρέπει να επιλέγεται τυχαία. Είναι προτιμότερο να χρησιμοποιούμε ως κλειδί το checksum του κώδικα. Με αυτόν τον τρόπο, η διαδικασία της αποκρυπτογράφησης θα αποτελεί ταυτόχρονα κι έναν έλεγχο για την αρτιότητα του κώδικα. Εφόσον η αποκρυπτογράφηση πραγματοποιείται με επιτυχία, γνωρίζουμε ότι το κλειδί (το checksum του κώδικα) παραμένει ανέπαφο κι άρα ο κώδικας δεν έχει δεχτεί καμία τροποποίηση. Ένα πρόσθετο μέτρο είναι η σταδιακή κρυπτογράφηση του κώδικα με διαφορετικά κλειδιά. Όμως ούτε κι αυτό αποτελεί πανάκεια.

Πολυμορφικός κώδικας

Αν ασχοληθεί μια ομάδα crackers με τον κώδικά μας, είναι σίγουρο ότι θα καταφέρει να τον αναλύσει πλήρως, να αντιστρέψει τη λειτουργία του και τελικά να δημιουργήσει ένα "generic patch", το οποίο θα καταργεί αυτόματα κάθε σύστημα προστασίας. Για να ξεφύγουμε από αυτόν τον κίνδυνο πρέπει να γράψουμε τον κώδικά μας με τέτοιο τρόπο, ώστε το patch που έχουν φτιάξει οι crackers να μην εφαρμόζεται σε όλα τα εκτελέσιμα. Αναρωτιέστε πώς θα το πετύχουμε αυτό; Γράφοντας πολυμορφικό κώδικα, όπως άλλωστε κάνουν και πολλοί δημιουργοί ιών! Οι πολυμορφικοί ιοί μεταλλάσσουν τον κώδικά τους κάθε φορά που εισέρχονται σ' ένα αρχείο. Έτσι, αν και η λειτουργικότητά τους παραμένει αναλλοίωτη, η μορφή τους αλλάζει κι ο εντοπισμός δεν μπορεί να πραγματοποιηθεί με την απλή αναζήτηση μιας συμβολοσειράς. Καλά όλα αυτά, θα πείτε, αλλά πώς θα γράψουμε τέτοιον κώδικα; Ο patcher με τον οποίο προστατεύουμε τα προγράμματά μας θα πρέπει να λαμβάνει τα εξής δύο μέτρα:

1. Να τοποθετεί τα σημαντικότερα τμήματά του σε διαφορετικές θέσεις, κάθε φορά που ενσωματώνεται σε ένα πρόγραμμα (εικόνα 1).
2. Να χρησιμοποιεί κάθε φορά διαφορετικό αλγόριθμο κρυπτογράφησης.

Μπορεί αυτά τα δύο μέτρα να μοιάζουν απλά, αλλά στην υλοποίηση θα διαπιστώσετε ότι δεν είναι. Για την επίτευξη του πρώτου, οι ρουτίνες που απαρτίζουν τον εμφυτεύσιμο κώδικα του patcher πρέπει να είναι κατάλληλα γραμμένες, ώστε να εκτελούνται κανονικά από οποιαδήποτε



Εικόνα 1
Οι ρουτίνες του patcher που θα ενσωματωθούν στο προστατευμένο εκτελέσιμο είναι κατάλληλα γραμμένες ώστε να εκτελούνται χωρίς πρόβλημα από οπουδήποτε. Έτσι, ο patcher έχει την ευχέρεια να τις γράψει με τυχαία σειρά κάθε φορά.

περιοχή της μνήμης. Εάν έχουμε λοιπόν πέντε τέτοιες ρουτίνες, μπορούμε να τις τοποθετούμε κάθε φορά με τυχαία σειρά. Με αυτόν τον τρόπο, το generic patch ενός cracker που προσπαθεί να αλλοιώσει συγκεκριμένες περιοχές των προστατευμένων προγραμμάτων θα αποτυγχάνει και κατά πάσα πιθανότητα θα καταστρέφει ολοσχερώς το εκτελέσιμο. Το δεύτερο μέτρο προστασίας, η χρήση διαφορετικών αλγόριθμων κρυπτογράφησης, συνεπάγεται ότι ο εμφυτευμένος κώδικας του patcher θα είναι κάθε φορά διαφορετικός. Επομένως, ένας generic patcher που αναζητά συγκεκριμένες αλληλουχίες bytes, θα αποτυγχάνει να βρει το στόχο του.

Αλγόριθμοι κρυπτογράφησης

Πιστεύουμε ότι τα παραπάνω είναι κατανοητά. Είναι πολύ πιθανό, όμως, να έχει γεννηθεί το εξής εύλογο ερώτημα: Πού θα βρούμε τόσους πολλούς διαφορετικούς αλγόριθμους κρυπτογράφησης, ώστε κάθε εκτελέσιμο που κλειδώνουμε να χρησιμοποιεί κάποιον διαφορετικό; Η απάντηση είναι απλή: Δεν θα τους βρούμε πουθενά! Αντίθετα, θα τους κατασκευάσουμε μόνοι μας ;)

Μια εξαιρετικά απλή μέθοδο κρυπτογράφησης αποτελεί η εφαρμογή της λογικής πράξης XOR. Ας υποθέσουμε, για παράδειγμα, ότι θέλουμε να κρυπτογραφήσουμε την τιμή 0x12345678. Αρχικά επιλέγουμε μια αυθαίρετη τιμή (π.χ., την 0xAABBCCDD) η οποία θα αποτελεί το κλειδί κρυπτογράφησης κι αργότερα της αποκρυπτογράφησης. Στη συνέχεια πραγματοποιούμε την πράξη XOR μεταξύ των δύο τιμών. Το αποτέλεσμα είναι 0xB88F9AA5 και, όπως βλέπετε, μοιάζει εντελώς άσχετο με την αρχική τιμή. Αν πάρουμε αυτή τη μέθοδο και την εμπλουτίσουμε με μερικές ακόμα πράξεις, μπορούμε να δημιουργήσουμε έναν ελαφρώς συνθετότερο αλγόριθμο κρυπτογράφησης, ο οποίος ωστόσο θα είναι άριστος για τη δουλειά μας. Σκεφτείτε την εξής απλή παραλλαγή: Προσθέτουμε στην αρχική τιμή το 0x11111111, κάνουμε XOR με την τιμή 0xAABBCCDD και μετά αφαιρούμε την τιμή 0x22222222. Ο παραπάνω αλγόριθμος σε κώδικα Assembly θα ήταν κάπως έτσι:

```
LODSD          ; φόρτωσε 4 bytes από τη μνήμη στον EAX
ADD        EAX, 0x11111111
XOR        EAX, 0xAABBCCDD
SUB        EAX, 0x22222222
STOSD          ; αποθήκευσε την τιμή του EAX στη μνήμη
```

Η αντίστοιχη ρουτίνα αποκρυπτογράφησης θα είχε ως εξής:

```
LODSD      ; φόρτωσε 4 bytes από τη μνήμη στον EAX
ADD  EAX, 0x22222222
XOR  EAX, 0xAABBCCDD
SUB  EAX, 0x11111111
STOSD      ; αποθήκευσε την τιμή του EAX στη μνήμη
```

Είναι φανερό ότι οι δύο ρουτίνες συγγενεύουν στενά. Οι πράξεις έχουν αντιστραφεί (η πρόσθεση έγινε αφαίρεση και η αφαίρεση έγινε πρόσθεση), ενώ έχουν τοποθετηθεί και με την αντίστροφη σειρά. Φυσικά, επειδή δεν θα κρυπτογραφούσαμε μόνο 4 bytes, η ρουτίνα κρυπτογράφησης θα μπορούσε να τοποθετηθεί σε ένα loop και να πάρει την ακόλουθη μορφή:

```
MOV  ECX, SizeOfBuffer   ; ο ECX αποτελεί το μετρητή των επαναλήψεων
myLoop1:
LODSD                    ; φόρτωσε 4 bytes από τη μνήμη στον EAX
ADD        EAX, 0x11111111
XOR        EAX, 0xAABBCCDD
SUB        EAX, 0x22222222
XOR        EAX, ECX      ; XOR με την τιμή του ECX ;-)
STOSD
LOOP       myLoop1       ; επανέλαβε μέχρι να μηδενιστεί ο ECX
```

Ακόμα και για αυτούς που δεν γνωρίζουν τη γλώσσα Assembly, νομίζουμε ότι ο κώδικας είναι κατανοητός. Μερικές εντολές που πραγματοποιούν απλές πράξεις (στην Assembly πάντα) είναι οι XOR, ADD, SUB, ROR και ROL. Για τη δημιουργία ενός νέου αλγορίθμου κρυπτογράφησης, αρκεί να αξιοποιήσουμε αυτές τις εντολές και να σχηματίσουμε μια τυχαία αλληλουχία πράξεων. Για την αποκρυπτογράφηση αρκεί να σχηματίσουμε την αντίστροφη αλληλουχία, χρησιμοποιώντας και τις "αντίστροφες" πράξεις.

Γεννήτρια αλγορίθμων κρυπτογράφησης

Τώρα που εξετάσαμε τη μορφή μερικών απλών αλγορίθμων κρυπτογράφησης και αποκρυπτογράφησης, θα δούμε έναν μηχανισμό που σχηματίζει τέτοιους αλγορίθμους δυναμικά. Ουσιαστικά, θα εξετάσουμε τη λειτουργία μιας γεννήτριας αλγορίθμων κρυπτογράφησης.

Ξεκινάμε καθορίζοντας το "μέγεθος" του αλγορίθμου: Το πλήθος των πράξεων που θα περιλαμβάνει. Για να δυσκολέψουμε κάπως τους crackers, θέλουμε οι αλγόριθμοι κρυπτογράφησης να περιλαμβάνουν ένα τυχαίο πλήθος πράξεων, από 5 έως και 15. Για το σκοπό αυτό χρησιμοποιούμε έναν τυχαίο αριθμό από το 0 έως το 10, στον οποίο προσθέτουμε το 5. Στη συνέχεια επιλέγουμε τυχαία μία από τις διαθέσιμες πράξεις. Αν πέσουμε στις εντολές ROR και ROL (ολίσθηση των bits προς τα δεξιά ή τα αριστερά), για την εκτέλεση της πράξης χρησιμοποιούμε και μια τυχαία τιμή από το 1 έως το 31 (επειδή τα bits σε κάθε word είναι 32). Αν επιλέξουμε κάποια άλλη εντολή, για την εκτέλεση της αντίστοιχης πράξης χρησιμοποιούμε μια τυχαία τιμή από το 1 έως το 0xFFFFFFFF. Δείτε έναν αλγόριθμο κρυπτογράφησης που θα μπορούσε να δημιουργηθεί με αυτή τη μέθοδο:

```
ADD        EAX, 0x17D248AA
XOR        EAX, 0x4132D69B
ROR        EAX, 7
SUB        EAX, 0x2245ADAB
ROL        EAX, 22
SUB        EAX, 0x00450AAD
```

Η ίδια μέθοδος θα μπορούσε να μας δώσει και έναν εντελώς διαφορετικό αλγόριθμο, όπως ο ακόλουθος:

```
ROR         EAX, 30
ADD         EAX, 0x48AAABDC
XOR         EAX, 0x4139B332
XOR         EAX, 0x43D9B454
ROL         EAX, 1
SUB         EAX, 0x2245ADAB
XOR         EAX, 0xB4BB1A32
ROL         EAX, 15
SUB         EAX, 0x014253DA
ROP         EAX, 8
ADD         EAX, 0x214181AB
```

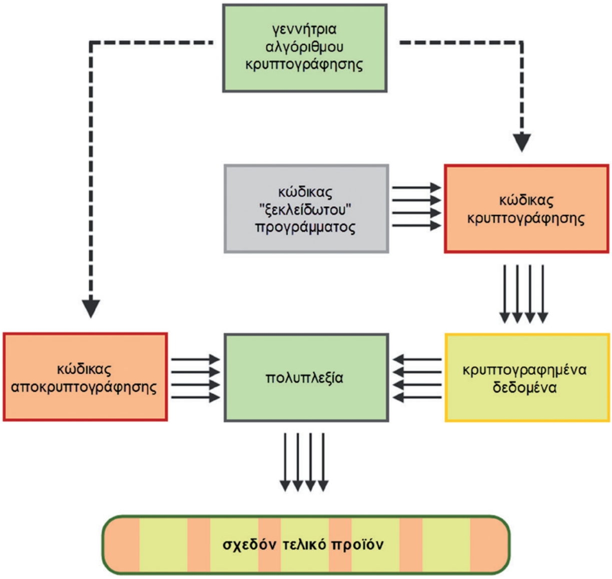
Όπως βλέπετε, ο κώδικας σε κάθε περίπτωση είναι εντελώς διαφορετικός. Φανταστείτε τώρα έναν patcher, ο οποίος υλοποιεί την παραπάνω μέθοδο, για την παραγωγή αλγορίθμων κρυπτογράφησης. Κάθε φορά που θα προστατεύαμε ένα πρόγραμμα, ο patcher θα σχημάτιζε μια ρουτίνα κρυπτογράφησης και μία αντίστοιχη για την αποκρυπτογράφηση. Με τη βοήθεια της πρώτης ρουτίνας ο patcher θα κρυπτογραφούσε τον κώδικα του προγράμματος που θέλουμε να κλειδώσουμε. Η δεύτερη ρουτίνα (αυτή που πραγματοποιεί την αποκρυπτογράφηση) θα προσαρτάτο στον κώδικα που εμφυτεύει (inject) ο patcher στο εκτελέσιμο. Όπως αντιλαμβάνεστε, ο εμφυτευμένος κώδικας θα είχε κάθε φορά διαφορετικό μέγεθος. Αν μάλιστα είχαμε υλοποιήσει και το μέτρο προστασίας που αναφέραμε αρχικά (αυτό της τυχαίας αναδιάταξης ορισμένων τμημάτων), ο εμφυτευμένος κώδικας δεν θα παρουσίαζε μόνο μια μεταβολή στο μέγεθος, αλλά είχε και εξ ολοκλήρου διαφορετική μορφή!

Εδώ πρέπει να σταθούμε σε μία σημαντική λεπτομέρεια. Ας υποθέσουμε ότι διαθέτουμε έναν patcher, ο οποίος παράγει δυναμικά αλγορίθμους κρυπτογράφησης. Ας υποθέσουμε επίσης ότι με αυτόν τον patcher κλειδώνουμε κάποιο πρόγραμμα. Προφανώς, αν δημιουργήσουμε αντίγραφα του προστατευμένου προγράμματος (π.χ., για τη διανομή του) θα είναι πανομοιότυπα μεταξύ τους. Διαφοροποίηση θα παρατηρήσουμε μόνον αν χρησιμοποιήσουμε τον patcher εκ νέου, για να κλειδώσουμε μερικά αντίγραφα του ξεκλείδωτου προγράμματος. Αυτός ο τρόπος εργασίας, όμως, δεν είναι καθόλου πρακτικός. Αν θέλουμε να εξασφαλίσουμε ότι *κάθε* αντίγραφο του κλειδωμένου προγράμματος θα έχει διαφορετική μορφή, πρέπει να μιμηθούμε τους πολυμορφικούς ιούς: Πρέπει ο patcher να εμφυτεύει τον κατάλληλο κώδικα, ο οποίος σε κάθε εκτέλεση του κλειδωμένου προγράμματος θα παράγει έναν νέο αλγόριθμο κρυπτογράφησης. Με αυτόν τον αλγόριθμο θα κρυπτογραφείται εξαρχής ο κώδικας του προγράμματος, ενώ στη συνέχεια θα εμφυτεύεται και η νέα ρουτίνα αποκρυπτογράφησης. Αυτή η συμπεριφορά θα ήταν αποτελεσματικότατη αλλά, όπως είπαμε ήδη, ομοιάζει με τον τρόπο δράσης των πολυμορφικών ιών. Ένα πρόγραμμα που λειτουργεί κατ' αυτόν τον τρόπο θα θορυβήσει αμέσως το εκάστοτε anti-virus, το οποίο με τη σειρά του θα απομονώσει το πρόγραμμα και θα απαγορεύσει την εκτέλεσή του :/

Μεγαλύτερη ποικιλομορφία

Όπως είπαμε, η διαρκής μεταβολή της μεθόδου κρυπτογράφησης και του εμφυτευμένου κώδικα σε κάθε εκτέλεση του προστατευμένου προγράμματος, δημιουργεί προβλήματα και δεν συνιστάται. Μπορούμε όμως να αυξήσουμε ακόμα περισσότερο

την ποικιλομορφία που πετυχαίνει ο patcher και ταυτόχρονα να δυσκολέψουμε το έργο του cracker ακόμα περισσότερο. Ένας τρόπος για να το πετύχουμε αυτό, είναι να πλέξουμε τα κρυπτογραφημένα δεδομένα με τον κώδικα που πραγματοποιεί την αποκρυπτογράφηση! Για το σκοπό αυτό αρκεί να γράψουμε μια απλή ρουτίνα, η οποία θα τεμαχίζει σε μικρά πακέτα τα κρυπτογραφημένα δεδομένα, καθώς και τον κώδικα της αποκρυπτογράφησης. Στη συνέχεια, η ίδια ρουτίνα θα τοποθετεί τα πακέτα των δύο ειδών εναλλάξ και σε διαδοχικές θέσεις μέσα στο κλειδωμένο πρόγραμμα. Υπό μία έννοια, η εν λόγω ρουτίνα θα πραγματοποιεί ένα είδος πολυπλεξίας[1]. Σε αυτή τη διαδικασία μπορούμε να εισάγουμε και τον παράγοντα της τύχης, ώστε κάθε φορά που χρησιμοποιούμε τον patcher να πραγματοποιείται ένα διαφορετικό είδος πλεξίματος. Για παράδειγμα, μπορούμε να μεταβάλλουμε τυχαία το μέγεθος των πακέτων ή ακόμα και να εισάγουμε ψεύτικα δεδομένα (dummy data) μέσα στα αληθινά. Αν κάνουμε κάτι τέτοιο, ο κώδικας του patcher θα πρέπει να αποθηκεύσει κάπου τα σχετικά μεγέθη, ώστε να είναι σε θέση να πραγματοποιήσει την αντίστροφη διαδικασία. Το όφελος από την προσθήκη τυχαίων ποσοτήτων dummy data είναι διπλό: Όχι μόνο συμβάλλει στη διαφοροποίηση του κώδικα (στην πολυμορφία), αλλά αυξάνει και τη σύγχυση του επιτιθέμενου. Όταν κατά την εκτέλεση ενός κλειδωμένου προγράμματος πραγματοποιηθεί η αντίστροφη διαδικασία (το ξέμπλεγμα και η αποκρυπτογράφηση), τα dummy data θα μοιάζουν με δεδομένα που είτε δεν αποκρυπτογραφήθηκαν σωστά, είτε δεν έχουν αποκρυπτογραφηθεί καθόλου. Αυτή τη λεπτομέρεια την αναφέρουμε μόνο και μόνο για να σας δείξουμε πώς πρέπει να σκέφτεστε, όταν κατασκευάζετε έναν patcher ή κάποιον άλλο μηχανισμό προστασίας. Εξάλλου, ο πολυμορφισμός, η κρυπτογράφηση κι όλα όσα έχουμε εξετάσει, υποτάσσονται σε έναν υπέρτατο στόχο: Στο να δυσκολέψουμε τη ζωή του cracker όσο περισσότερο γίνεται!



Εικόνα 2
Ο αλγόριθμος κρυπτογράφησης τροφοδοτείται με τον κώδικα του προγράμματος που θέλουμε να προστατεύσουμε. Το προϊόν της κρυπτογράφησης πολυπλέκεται με τον αλγόριθμο αποκρυπτογράφησης. Σε αυτό το "μίγμα" θα προστεθούν κάποιες ρουτίνες του patcher, που αναλαμβάνουν την αποπολύπλεξη, την εκκίνηση της αποκρυπτογράφησης κ.ά.

1. Στα τηλεπικοινωνιακά συστήματα, πολυπλεξία (multiplexing) ονομάζεται η μέθοδος με την οποία διαφορετικά σήματα μεταδίδονται μέσα από ένα μόνο κανάλι. Το πιο γνωστό είδος πολυπλεξίας είναι το TDM (Time-Division Multiplexing), στο οποίο τα επιμέρους σήματα χρησιμοποιούν το κανάλι εκ περιτροπής, για προκαθορισμένα χρονικά διαστήματα.

Μ' αυτά και με τ' άλλα, όμως, είναι πιθανό να προκαλέσαμε σύγχυση και στη δική σας σκέψη. Ας ξαναδούμε λοιπόν τις εργασίες που πρέπει να πραγματοποιεί ένας patcher, για να παράγει πολυμορφικό κώδικα:

- Δημιουργία ενός αλγορίθμου κρυπτογράφησης με τυχαίο μέγεθος (πλήθος πράξεων).
- Δημιουργία του αντίστοιχου αλγορίθμου αποκρυπτογράφησης.
- Κρυπτογράφηση του κώδικα του προγράμματος που επιθυμούμε να προστατεύσουμε.
- Πολυπλεξία των κρυπτογραφημένων δεδομένων με τον κώδικα που πραγματοποιεί την αποκρυπτογράφηση.
- Εμφύτευση των ρουτινών που θα πραγματοποιούν το ξέμπλεγμα και θα εκτελούν τον κώδικα αποκρυπτογράφησης. Οι εν λόγω ρουτίνες πρέπει να τοποθετούνται σε τυχαίες θέσεις!

Σ' αυτό το σημείο θα περίμενε κανείς την παράθεση ενός ολοκληρωμένου προγράμματος, που πετυχαίνει όσα περιγράψαμε. Νομίζουμε όμως ότι θα ωφεληθείτε *πολύ* περισσότερο αν αφομοιώσετε τη θεωρία και προσπαθήσετε να γράψετε τον κώδικα μόνοι σας. Αυτό βέβαια δεν σημαίνει ότι θα σας αφήσουμε να κάνετε και όλη τη... λάντζα. Για να σας απαλλάξουμε από ένα ανιαρό κομμάτι της εργασίας και ταυτόχρονα να σας ανοίξουμε την όρεξη, θα σας δώσουμε μερικούς πίνακες που διευκολύνουν την κατασκευή μιας γεννήτριας αλγορίθμων κρυπτογράφησης. Ο πρώτος πίνακας (MyOpCodes) περιλαμβάνει εντολές που θα μπορούσαμε να χρησιμοποιήσουμε σε έναν απλό αλγόριθμο κρυπτογράφησης:

```
MyOpCodes[0][0] = 0x05;        // Add EAx
MyOpCodes[0][1] = 0x81C1;      // Add Ecx
MyOpCodes[0][2] = 0x81C2;      // Add EDx
MyOpCodes[0][3] = 0x81C3;      // Add EBx
MyOpCodes[0][4] = 0x81C5;      // Add EBp
MyOpCodes[0][5] = 0x81C6;      // Add ESi
MyOpCodes[0][6] = 0x81C7;      // Add EDi

MyOpCodes[1][0] = 0x2D;        // Sub EAx
MyOpCodes[1][1] = 0x81E9;      // Sub Ecx
MyOpCodes[1][2] = 0x81EA;      // Sub EDx
MyOpCodes[1][3] = 0x81EB;      // Sub EBx
MyOpCodes[1][4] = 0x81ED;      // Sub EBp
MyOpCodes[1][5] = 0x81EE;      // Sub ESi
MyOpCodes[1][6] = 0x81EF;      // Sub EDi

MyOpCodes[2][0] = 0xC1C8;      // Ror EAx
MyOpCodes[2][1] = 0xC1C9;      // Ror Ecx
MyOpCodes[2][2] = 0xC1CA;      // Ror EDx
MyOpCodes[2][3] = 0xC1CB;      // Ror EBx
MyOpCodes[2][4] = 0xC1CD;      // Ror EBp
```

```
    MyOpCodes[2][5] = 0xC1CE;      // Ror ESi
    MyOpCodes[2][6] = 0xC1CF;      // Ror EDi

    MyOpCodes[3][0] = 0xC1C0;      // Rol EAx
    MyOpCodes[3][1] = 0xC1C1;      // Rol Ecx
    MyOpCodes[3][2] = 0xC1C2;      // Rol EDx
    MyOpCodes[3][3] = 0xC1C3;      // Rol EBx
    MyOpCodes[3][4] = 0xC1C5;      // Rol EBp
    MyOpCodes[3][5] = 0xC1C6;      // Rol ESi
    MyOpCodes[3][6] = 0xC1C7;      // Rol EDi

    MyOpCodes[4][0] = 0x35;        // Xor EAx
    MyOpCodes[4][1] = 0x81F1;      // Xor Ecx
    MyOpCodes[4][2] = 0x81F2;      // Xor EDx
    MyOpCodes[4][3] = 0x81F3;      // Xor EBx
    MyOpCodes[4][4] = 0x81F5;      // Xor EBp
    MyOpCodes[4][5] = 0x81F6;      // Xor ESi
    MyOpCodes[4][6] = 0x81F7;      // Xor EDi
```

Ο πίνακας MyMoves περιέχει την εντολή MOV, με όλους τους πιθανούς συνδυασμούς καταχωρητών. Η συγκεκριμένη εντολή μετακινεί δεδομένα από καταχωρητή σε καταχωρητή. Σας φαίνεται παράξενο που φτιάξαμε έναν πίνακα γι' αυτήν; Στους "χειροποίητους" αλγορίθμους κρυπτογράφησης, εκτός από την τυχαία χρήση διαφόρων πράξεων συνηθίζουμε να μετακινούμε τα δεδομένα από 'δώ κι από 'κεί. Αυτό το κάνουμε ακόμα κι όταν δεν υπάρχει ιδιαίτερος λόγος. Έτσι αυξάνουμε τη σύγχυση του cracker και ταυτόχρονα εμπλουτίζουμε την πολυμορφικότητα του κώδικα.

```
    MyMoves[0][0] = 0;             // Mov EAx, EAx (Δεν υπάρχει)
    MyMoves[0][1] = 0x8BC1;        // Mov EAx, ECx
    MyMoves[0][2] = 0x8BC2;        // Mov EAx, EDx
    MyMoves[0][3] = 0x8BC3;        // Mov EAx, EBx
    MyMoves[0][4] = 0x8BC5;        // Mov EAx, EBp
    MyMoves[0][5] = 0x8BC6;        // Mov EAx, ESi
    MyMoves[0][6] = 0x8BC7;        // Mov EAx, EDi

    MyMoves[1][0] = 0x8BC8;        // Mov ECx, EAx
    MyMoves[1][1] = 0;             // Mov ECx, ECx (Δεν υπάρχει)
    MyMoves[1][2] = 0x8BCA;        // Mov ECx, EDx
    MyMoves[1][3] = 0x8BCB;        // Mov ECx, EBx
    MyMoves[1][4] = 0x8BCD;        // Mov ECx, EBp
    MyMoves[1][5] = 0x8BCE;        // Mov ECx, ESi
    MyMoves[1][6] = 0x8BCF;        // Mov ECx, EDi

    MyMoves[2][0] = 0x8BD0;        // Mov EDx, EAx
```

```
    MyMoves[2][1] = 0x8BD1;        // Mov EDx, ECx
    MyMoves[2][2] = 0;             // Mov EDx, EDx (Δεν υπάρχει)
    MyMoves[2][3] = 0x8BD3;        // Mov EDx, EBx
    MyMoves[2][4] = 0x8BD5;        // Mov EDx, EBp
    MyMoves[2][5] = 0x8BD6;        // Mov EDx, ESi
    MyMoves[2][6] = 0x8BD7;        // Mov EDx, EDi

    MyMoves[3][0] = 0x8BD8;        // Mov EBx, EAx
    MyMoves[3][1] = 0x8BD9;        // Mov EBx, ECx
    MyMoves[3][2] = 0x8BDA;        // Mov EBx, EDx
    MyMoves[3][3] = 0;             // Mov EBx, EBx (Δεν υπάρχει)
    MyMoves[3][4] = 0x8BDD;        // Mov EBx, EBp
    MyMoves[3][5] = 0x8BDE;        // Mov EBx, ESi
    MyMoves[3][6] = 0x8BDF;        // Mov EBx, EDi

    MyMoves[4][0] = 0x8BE8;        // Mov EBp, EAx
    MyMoves[4][1] = 0x8BE9;        // Mov EBp, ECx
    MyMoves[4][2] = 0x8BEA;        // Mov EBp, EDx
    MyMoves[4][3] = 0x8BEB;        // Mov EBp, EBx
    MyMoves[4][4] = 0;             // Mov EBp, EBp (Δεν υπάρχει)
    MyMoves[4][5] = 0x8BEE;        // Mov EBp, ESi
    MyMoves[4][6] = 0x8BEF;        // Mov EBp, EDi

    MyMoves[5][0] = 0x8BF0;        // Mov ESi, EAx
    MyMoves[5][1] = 0x8BF1;        // Mov ESi, ECx
    MyMoves[5][2] = 0x8BF2;        // Mov ESi, EDx
    MyMoves[5][3] = 0x8BF3;        // Mov ESi, EBx
    MyMoves[5][4] = 0x8BF5;        // Mov ESi, EBp
    MyMoves[5][5] = 0;             // Mov ESi, ESi (Δεν υπάρχει)
    MyMoves[5][6] = 0x8BF7;        // Mov ESi, EDi

    MyMoves[6][0] = 0x8BF8;        // Mov EDi, EAx
    MyMoves[6][1] = 0x8BF9;        // Mov EDi, ECx
    MyMoves[6][2] = 0x8BFA;        // Mov EDi, EDx
    MyMoves[6][3] = 0x8BFB;        // Mov EDi, EBx
    MyMoves[6][4] = 0x8BFD;        // Mov EDi, EBp
    MyMoves[6][5] = 0x8BFE;        // Mov EDi, ESi
    MyMoves[6][6] = 0;             // Mov EDi, EDi (Δεν υπάρχει)
```

Στο ξεκίνημα της αποκρυπτογράφησης και πριν αρχίσουμε να μετακινούμε τις τιμές των καταχωρητών, θα ήταν φρόνιμο να αποθηκεύσουμε τις τιμές τους στη στοίβα. Όταν ολοκληρωθεί η διαδικασία, θα χρειαστεί να πάρουμε τις αποθηκευμένες τιμές από τη στοίβα, ώστε όλοι οι καταχωρητές να επανέλθουν στην αρχική τους κατάσταση. Για αυτές τις δύο εργασίες θα χρειαστούμε τις εντολές PUSH (προσθήκη στη στοίβα) και POP (λήψη από τη στοίβα). Οι πίνακες MyPush και MyPop περιλαμβάνουν τους πιθανούς συνδυασμούς αυτών των εντολών με τους διάφορους καταχωρητές...

```
MyPush[0] = 0x50;     // Push EAx
MyPush[1] = 0x51;     // Push ECx
MyPush[2] = 0x52;     // Push EDx
MyPush[3] = 0x53;     // Push EBx
MyPush[4] = 0x55;     // Push EBp
MyPush[5] = 0x56;     // Push ESi
MyPush[6] = 0x57;     // Push Edi


MyPop[0] = 0x58;      // Pop EAx
MyPop[1] = 0x59;      // Pop ECx
MyPop[2] = 0x5A;      // Pop EDx
MyPop[3] = 0x5B;      // Pop EBx
MyPop[4] = 0x5D;      // Pop EBp
MyPop[5] = 0x5E;      // Pop ESi
MyPop[6] = 0x5F;      // Pop EDi
```

Όπως καταλαβαίνετε, οι πίνακες που παραθέσαμε περιλαμβάνουν το OpCode (τον κώδικα σε γλώσσα μηχανής) της εκάστοτε εντολής. Η χρήση αυτών των πινάκων είναι σχεδόν αναγκαία για το πρόγραμμα που θέλουμε να γράψουμε. Σκεφτείτε ότι ο κώδικάς μας θα σχηματίζει… κώδικα! Για να το πετύχουμε αυτό, ο κώδικάς μας θα πρέπει να γνωρίζει τα OpCode όλων των εντολών που ενδέχεται να χρησιμοποιηθούν.

Με εφόδιο τη θεωρία και τους παραπάνω πίνακες, ελπίζουμε να περάσετε αρκετές ευχάριστες ώρες γράφοντας κώδικα! Εμείς σας ευχόμαστε καλή επιτυχία και περιμένουμε να δούμε τα αποτελέσματα της δουλειάς σας ή και να ακούσουμε τις απορίες σας.



Skill: Beginner
Tags: Android, x86, port, dual boot, EasyBCD, Ext2Fsd



by multiPetros

Ένα πράσινο ρομπότ για το PC σας [μέρος 2]

Ακολουθώντας τις οδηγίες του πρώτου μέρους του μικρού αφιερώματός μας, που αρχίζει από τη σελίδα 34, έχουμε βρεθεί με το Android x86 εγκατεστημένο στον υπολογιστή μας. Προς το παρόν όμως δεν μπορούμε να κάνουμε κάτι μ' αυτό, αφού τίποτα δεν μαρτυρά την ύπαρξή του. Στο δεύτερο αυτό μέρος θα εγκαταστήσουμε τον boot loader για το Android μας και θα ρυθμίσουμε κατάλληλα το μενού εκκίνησης. Επιπλέον, μετά την είσοδό μας στο φρεσκοεγκατεστημένο λειτουργικό σύστημα, θα πραγματοποιήσουμε μια σειρά ρυθμίσεων που θα μας βοηθήσουν να το αξιοποιήσουμε στο έπακρο.

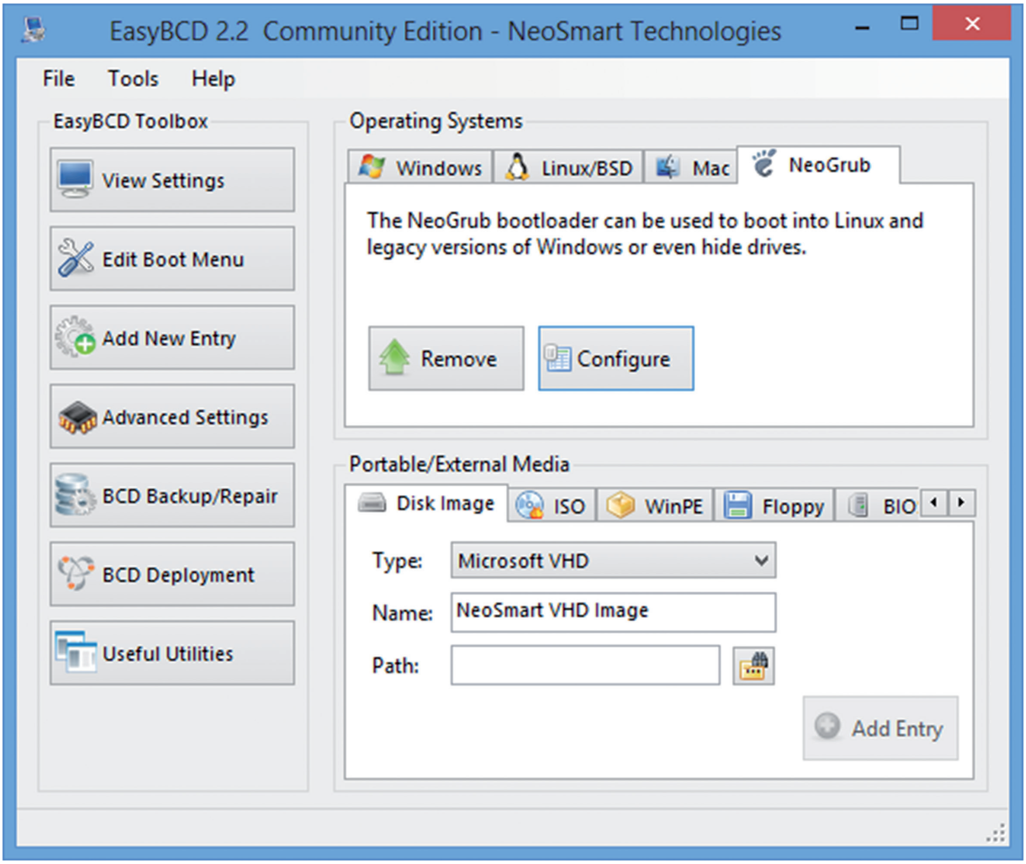
Μετά από μια επιτυχή εγκατάσταση του Android x86 στο PC μας και έχοντας επιλέξει να μη γίνει εγκατάσταση του boot manager από το ίδιο το Android x86, επανεκκινώντας τον υπολογιστή μας θα εισέλθουμε κανονικά στο κύριο λειτουργικό σύστημα, σαν το νέο λειτουργικό να μην εγκαταστάθηκε ποτέ. Αυτό είναι πολύ λογικό, αφού ακόμα δεν έχουμε ενημερώσει τον boot manager που ήδη έχουμε για τις αλλαγές που πραγματοποιήσαμε. Όπως και πριν, θα εξετάσουμε τόσο την περίπτωση του συστήματος με εγκατεστημένη κάποια διανομή Linux, όσο και αυτή των Windows. Στη δεύτερη περίπτωση, από την οποία και θα ξεκινήσουμε, θα δώσουμε λίγο περισσότερη έμφαση, θεωρώντας ότι οι χρήστες του Linux είναι περισσότερο εξοικειωμένοι με το σύστημά τους, αλλά και λόγω της ποικιλομορφίας των λύσεων που μπορούν να εφαρμοστούν σ' αυτή την περίπτωση.

Εξετάζοντας λοιπόν την περίπτωση με τα Windows, το εύρος των επιλογών μας είναι σαφώς μικρότερο. Ο boot manager των Windows, από τις εκδόσεις Vista κι έπειτα, λαμβάνει όλες τις απαραίτητες πληροφορίες για την εκκίνηση του συστήματος συμβουλευόμενος μια βάση δεδομένων με όνομα BCD (Boot Configuration Data). Τα Windows είναι εφοδιασμένα με ένα εργαλείο της γραμμής εντολών που ακούει στο όνομα bcdedit. Εμείς, όμως, θα παραβλέψουμε επιδεικτικά την ύπαρξή του και θα αναζητήσουμε μια ευκολότερη και φυσικά πιο visual λύση. Τρία δημοφιλή εργαλεία γι' αυτή τη δουλειά είναι τα Visual BCD Editor, BOOTICE και EasyBCD. Απ' αυτά εμείς επιλέξαμε το EasyBCD, το οποίο βρήκαμε ιδιαιτέρως εύχρηστο, με πολλές δυνατότητες παραμετροποίησης. Ο κύριος λόγος που το επιλέξαμε είναι ο NeoGrub boot manager που φέρει μαζί του. Μπορείτε να κατεβάσετε το EasyBCD στον υπολογιστή σας, από τη διεύθυνση http://neosmart.net/EasyBCD. Η βασική του έκδοση, η Community Edition, όπως τιτλοφορείται, είναι δωρεάν για προσωπική χρήση.

Μετά την εγκατάσταση του EasyBCD στον υπολογιστή μας και την εκτέλεση του προγράμματος, από τα κουμπιά που εμφανίζονται στο "Toolbox", στ' αριστερά του παραθύρου, επιλέγουμε και πατάμε το "Add New Entry", έτσι ώστε να δημιουργήσουμε μια νέα εγγραφή στο boot menu. Στην περιοχή "Operating Systems" διαλέγουμε την καρτέλα "NeoGrub" κι απλώς πατάμε στο κουμπί "Install". Μόλις η εγκατάστασή του και η ενημέρωση του BCD των Windows έχουν ολοκληρωθεί, ενεργοποιείται το κουμπί "Configure". Πατώντας αυτό το κουμπί ανοίγει το Notepad, παρέχοντάς μας τη δυνατότητα παραμετροποίησης των εγγραφών του NeoGrub. Η σύνταξη του αρχείου ακολουθεί τους γενικούς κανόνες της οικογένειας των Grub Boot Managers.

Προτού όμως προχωρήσουμε σε οποιαδήποτε ρύθμιση θα πρέπει να γνωρίζουμε ορισμένες πληροφορίες γύρω από την εγκατάσταση του Android x86. Πρώτον, το δίσκο και το διαμέρισμα που έχει γίνει η εγκατάσταση. Δεύτερον, το όνομα της συσκευής στην οποία βρίσκεται εγκατεστημένο. Και τρίτον, τη διαδρομή των αρχείων kernel και initrd.img.

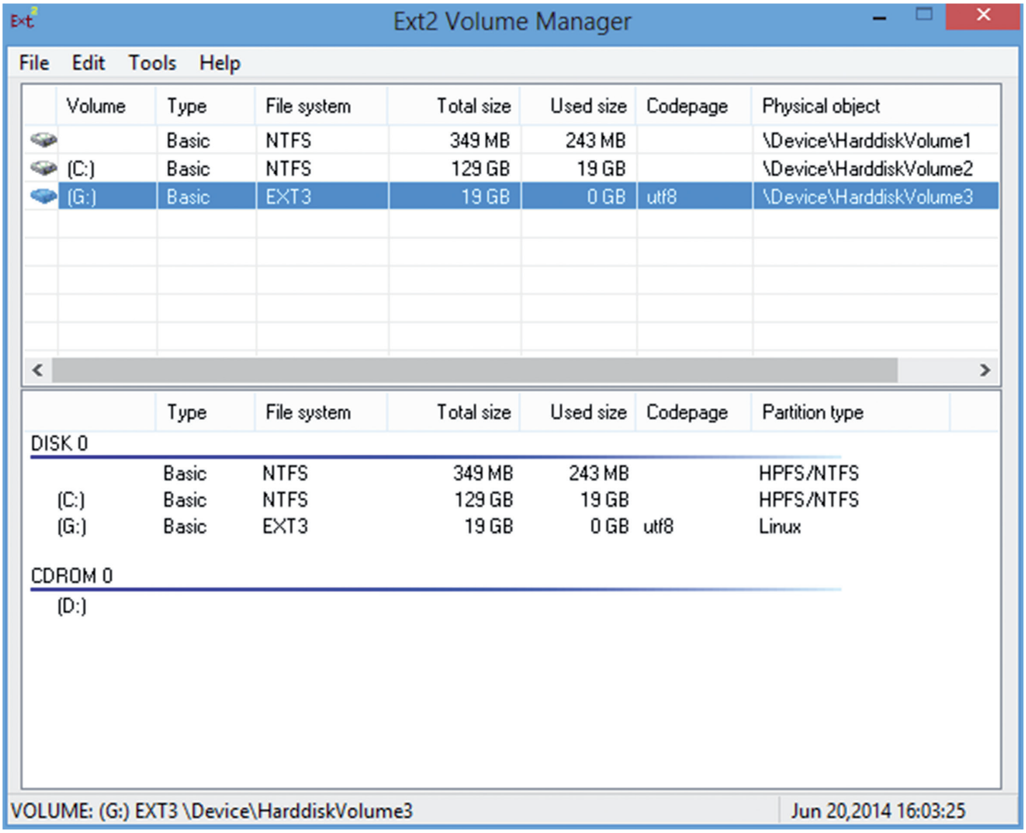
Για το πρώτο, αρκεί να θυμόμαστε πού κάναμε την εγκατάσταση. Στο δικό μας σύστημα, για παράδειγμα, αυτή πραγματοποιήθηκε στο τρίτο διαμέρισμα (στο πρώτο, με μέγεθος περί τα 100MB, βρίσκεται ο boot loader των Windows και στο δεύτερο τα ίδια τα Windows) του πρώτου δίσκου (έτσι κι αλλιώς δεν είχαμε άλλον). Στον τρόπο σύνταξης του αρχείου ρυθμίσεων του Grub, οι δίσκοι του συστήματος ξεκινούν με τα γράμματα hd και ακολουθεί η αρίθμηση τους, ξεκινώντας από το μηδέν. Επίσης από το μηδέν ξεκινάει και η αρίθμηση των partitions του εκάστοτε δίσκου. Για τη δική μας λοιπόν εγκατάσταση, η τοποθεσία εγκατάστασης του Android x86 μεταφράζεται ως (hd0,2).



Το περιβάλλον εργασίας του EasyBCD, απ' όπου θα εγκαταστήσουμε και θα ρυθμίσουμε τον NeoGrub Boot Loader.

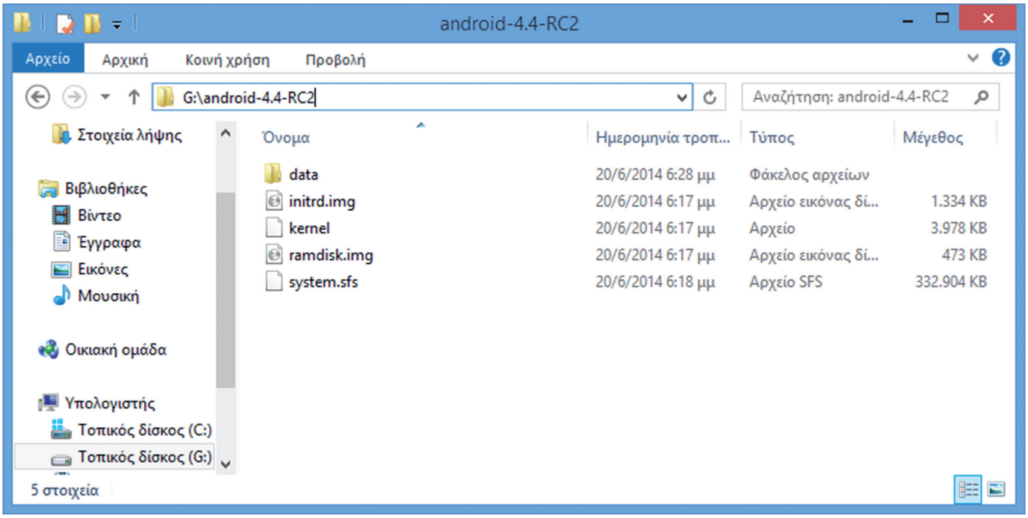
Το όνομα της συσκευής, τώρα, ακολουθεί τη γενική ονοματολογία του Linux, όπου οι δίσκοι γενικώς βρίσκονται στη διαδρομή /dev/sdXY, όπου X ένα γράμμα του λατινικού αλφαβήτου (ξεκινώντας από το a για τον πρώτο δίσκο) και όπου Y είναι ένας αριθμός (ξεκινώντας από το 0 για το πρώτο σε σειρά διαμέρισμα του δίσκου). Στην προκειμένη περίπτωση, μιλάμε για τη συσκευή /dev/sda2.

Η τοποθεσία που βρίσκονται τοποθετημένα τα αρχεία kernel και initrd.img απαιτεί λίγο παραπάνω ψάξιμο. Μπορούμε, π.χ., να ξεκινήσουμε το σύστημά μας με μια Live διανομή Linux και να περιηγηθούμε στα περιεχόμενα των κατατμήσεων του δίσκου μας, ανακαλύπτοντας τη διαδρομή των αρχείων αυτών. Εναλλακτικά, είναι δυνατόν να κάνουμε τα Windows να δουν τα περιεχόμενα του διαμορφωμένου κατά Ext3 διαμερίσματος που φιλοξενεί τα αρχεία του Android x86. Εμείς επιλέξαμε να κάνουμε το δεύτερο, καθώς αν καταφέρουμε να αποκτήσουμε πρόσβαση στα αρχεία του Android μέσα από τα Windows τότε η ευελιξία του dual boot συστήματός μας θα αναβαθμιστεί σημαντικά. Φανταστείτε, π.χ., ότι θα μπορούμε να ξεκινάμε μια δουλειά (ας πούμε τη δημιουργία ενός εγγράφου) μέσα από το περιβάλλον του Android και θα έχουμε τη δυνατότητα να τη συνεχίζουμε αργότερα μέσα από τα Windows, χωρίς



Με τη βοήθεια του Ext2Fsd θα προσαρτήσουμε και θα αποκτήσουμε πρόσβαση στα περιεχόμενα του Ext3 διαμερίσματος που φιλοξενεί το Android x86, μέσα από τα Windows.

να παρεμβάλλεται κάποιο εξωτερικό μέσο αποθήκευσης ή κάποια υπηρεσία cloud. Για να μην πούμε και για τις άπειρες δοκιμές και πειραματισμούς που θα μπορούμε με ευκολία να πραγματοποιήσουμε στο Android ;) Γι' αυτούς λοιπόν και παρόμοιους λόγους, επιστρατεύουμε το εργαλείο Ext2Fsd (Ext2 File System Driver). Πρόκειται για έναν driver που επιτρέπει στα Windows να έχουν πρόσβαση για ανάγνωση κι εγγραφή σε διαμερίσματα με συστήματα αρχείων Ext2, Ext3 και Ext4. Πέρα από τον driver, το Ext2Fsd προσφέρει κι ένα εργαλείο για το περιβάλλον γραφικών που μας επιτρέπει να προσαρτούμε τόμους του συστήματος που θέλουμε να είναι ορατοί στα Windows. Η εφαρμογή είναι προϊόν Ελεύθερου Λογισμικού και μπορούμε να κατεβάσουμε την τελευταία έκδοση της από τη διεύθυνση `www.ext2fsd.com`. Αφού λοιπόν εγκαταστήσουμε το Ext2Fsd, ανοίγουμε το πρόγραμμα κι από το κεντρικό του παράθυρο μπορούμε να βρούμε το διαμέρισμα όπου έχουμε εγκαταστήσει το Android x86. Με διπλό κλικ πάνω στην εγγραφή του εμφανίζεται ένα δεύτερο παράθυρο, απ' όπου μπορούμε να επιλέξουμε τις ρυθμίσεις για την προσάρτηση του διαμερίσματος. Για να ενεργοποιήσουμε την προσάρτηση τσεκάρουμε το "Automatically mount via Ext2Mgr" και ακριβώς δίπλα επιλέγουμε το γράμμα που θέλουμε να αντιστοιχίσουμε



Το διαμέρισμα του Android προσαρτήθηκε και του αποδόθηκε το γράμμα G:. Μέσα από τη διαχείριση αρχείων των Windows, μπορούμε να περιηγηθούμε στα περιεχόμενά του και να εντοπίσουμε τα αρχεία που μας ενδιαφέρουν.

στο διαμέρισμα. Ακριβώς από πάνω υπάρχει τσεκαρισμένη η επιλογή "Mount volume in readonly mode", την οποία αποεπιλέγουμε αν επιθυμούμε να αποκτήσουμε και δυνατότητα εγγραφής στο διαμέρισμα αυτό. Όταν είμαστε έτοιμοι πατάμε στο "Apply", κλείνουμε το παράθυρο του προγράμματος και μεταβαίνουμε στο διαχειριστή αρχείων των Windows. Εκεί, θα διαπιστώσουμε ότι ένας νέος δίσκος έχει κάνει την εμφάνισή του. Στη ρίζα αυτού υπάρχουν δύο φάκελοι: Ο ένας είναι ο γνωστός από τις διάφορες διανομές του Linux "lost+found" και ο άλλος τιτλοφορείται "android-4.4-RC2". Ανοίγοντας τον δεύτερο, με χαρά θα διαπιστώσουμε ότι είναι αυτός που περιέχει τα αρχεία που ψάχνουμε. Σημειώνουμε το όνομα του φακέλου κι έτσι η αναζήτησή μας τελειώνει κάπου εδώ.

Ερχόμαστε πάλι πίσω στο EasyBCD, και ανοίγουμε το αρχείο ρυθμίσεων του NeoGrub. Σε αυτό, και με τις πληροφορίες που καταφέραμε να συλλέξουμε, συμπληρώνουμε τις εξής γραμμές:

```
default=0

timeout=0

title    Android-x86 4.4

root     (hd0,2)

kernel   /android-4.4-RC2/kernel quiet root=/dev/sda2 androidboot.
hardware=android_x86 pcie_aspm=force usbcore.autosuspend=2 video=-16
SRC=/android-4.4-RC2 quiet

initrd   /android-4.4-RC2/initrd.img
```

Τώρα βέβαια, μια και έχουμε ανοιχτό το EasyBCD, μπορούμε αν το επιθυμούμε να κάνουμε και μερικές ακόμα αλλαγές στο boot menu. Από τα κουμπιά του Toolbox, πατώντας το "Edit Boot Menu", μας παρέχεται η δυνατότητα αλλαγής της σειράς με την οποία θα εμφανίζονται οι εγγραφές του μενού εκκίνησης, η ρύθμιση του προεπιλεγμένου λειτουργικού, καθώς κι ο χρόνος αναμονής μέχρι την αυτόματη εκκίνηση του προεπιλεγμένου λειτουργικού. Με ένα κλικ στο κουμπί "Save Settings" αποθηκεύουμε και αυτές τις ρυθμίσεις.

Στην περίπτωση τώρα που στο σύστημά μας υπάρχει ως βασικό λειτουργικό σύστημα μια διανομή Linux με τον αρκετά διαδεδομένο boot loader Grub2, τα βήματα είναι σαφώς λιγότερα. Αυτό που πρέπει να κάνουμε ώστε να εμφανίζεται το Android x86, είναι να ενημερώσουμε τον Grub2 σχετικά με την ύπαρξή του. Αρχικά, όπως και πριν, θα πρέπει να θυμηθούμε ή να βρούμε σε ποιο partition έχουμε εγκαταστήσει το Android x86. Έστω ότι εγκαταστάθηκε στο /dev/sda2, το οποίο αντιστοιχεί στο (hd0,2) του Grub. Κάνοντας mount το διαμέρισμα και ρίχνουντας μια ματιά στη δομή των φακέλων του,

```
mount -t ext3 /dev/sda2 /media/androidhouse

ls –la /media/androidhouse
```

μπορούμε να διαπιστώσουμε την τοποθεσία που βρίσκονται τα αρχεία kernel και initrd.img. Αυτά, πιθανότατα θα είναι σε κάποιον φάκελο /android-4.4-RC2. Με τον αγαπημένο μας editor και εφοδιασμένοι με δικαιώματα υπερχρήστη, ανοίγουμε τώρα το αρχείο /etc/grub.d/40_custom και στο τέλος του προσθέτουμε την εγγραφή για την εκκίνηση του Android x86, με σύνταξη σχεδόν πανομοιότυπη με αυτή που χρησιμοποιήσαμε για τον NeoGrub:

```
menuentry "Android" {

set root=(hd0,2)

linux /android-4.4-RC2/kernel quiet root=/dev/sda2 androidboot.
hardware=android_x86 SRC=/android-4.4-RC2

initrd /android-4.4-RC2/initrd.img

}
```

Τέλος, δεν πρέπει να ξεχάσουμε να ενημερώσουμε τον Grub γι' αυτές τις αλλαγές εκτελώντας ένα:

```
sudo update-grub
```

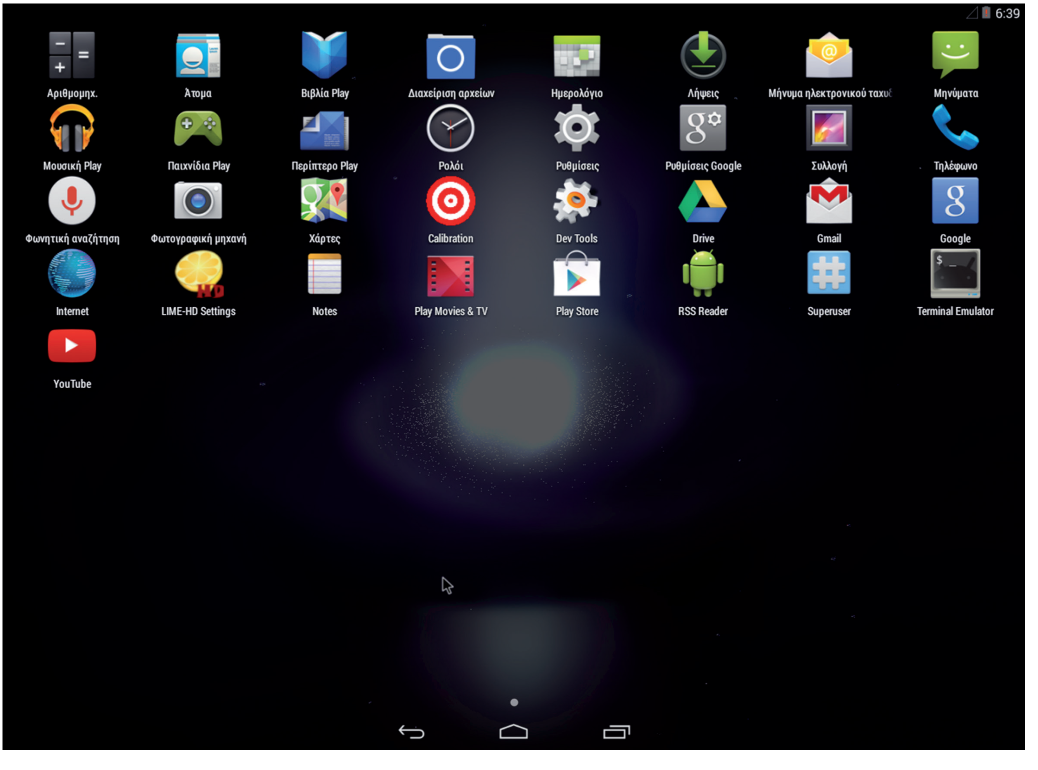
Τόσο οι εγγραφές στο Grub menu, όσο και η όλη διαδικασία, μπορεί να διαφέρουν λίγο ανάλογα με τη διαμόρφωση του δικού σας συστήματος. Η γενική πάντως κατεύθυνση είναι ακριβώς αυτή που σας περιγράφουμε. Αν φυσικά χρησιμοποιείτε κάποιον διαφορετικό boot manager, θα πρέπει να συμβουλευτείτε τις οδηγίες χρήσης του.

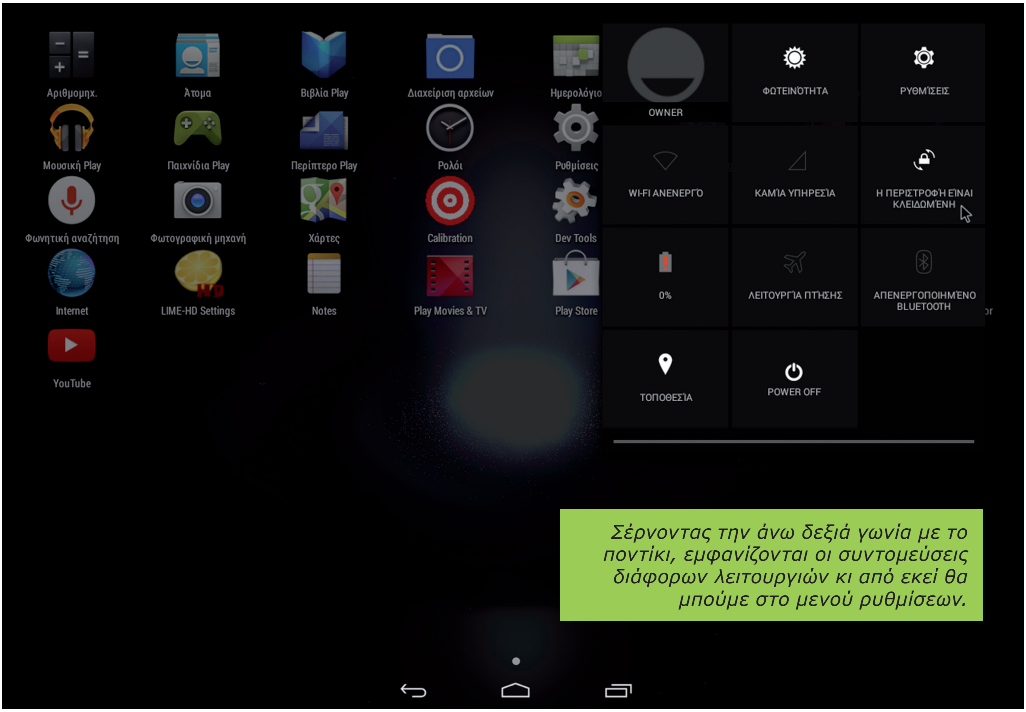
Είμαστε πλέον έτοιμοι! Μπορούμε να επανεκκινήσουμε τον υπολογιστή μας κι από το μενού του boot manager να επιλέξουμε την εκκίνηση του Android x86.

Το νέο μας λειτουργικό

Η πρώτη εκκίνηση του νέου μας λειτουργικού θα χρειαστεί λίγο περισσότερο χρόνο από το κανονικό. Το σύστημα θα αρχικοποιηθεί, εξετάζοντας τις συσκευές του υπολογιστή κι εκτελώντας τις απαραίτητες ρυθμίσεις. Αν είστε εξοικειωμένοι με το Android, θα γνωρίζετε τότε ότι στο πρώτο ξεκίνημα θα μας ζητηθεί ένας λογαριασμός Google. Για όσους αναρωτιούνται γιατί, να πούμε ότι ο λογαριασμός αυτός είναι απαραίτητος για την εγκατάσταση εφαρμογών από το Google Play, αλλά και για τη χρήση των σχετικών υπηρεσιών κι εφαρμογών της Google. Αφού ολοκληρώσουμε κι αυτό το βήμα, το Android μάς καλωσορίζει με μερικές βασικές οδηγίες για τη χρήση του περιβάλλοντός του. Να σημειώσουμε ότι η έκδοση 4.4 που κατεβάσαμε, γνωστή και με το κωδικό όνομα Kit-Kat, είναι η τελευταία έκδοση του δημοφιλούς mobile λειτουργικού. Ίσως λοιπόν συναντήσετε κάποιες μικροδιαφορές σε σχέση με αυτή που βρίσκεται εγκατεστημένη στο κινητό σας τηλέφωνο ή tablet. Οι προγραμματιστές του Android x86 έχουν δανειστεί μερικά στοιχεία από την πολύ δημοφιλή custom διανομή Cyanogen. Ανάμεσα σ' αυτά βρίσκεται και ο προεγκατεστημένος file manager, τον οποίο βέβαια μπορείτε να αντικαταστήσετε με κάποιον της επιλογής σας.

Περιττό να πούμε ότι εξαρχής έχετε root πρόσβαση στο σύστημα. Επομένως μπορείτε να κάνετε όσες παραμετροποιήσεις και πειραματισμούς τραβά η όρεξή σας! Την πρόσβαση root μπορείτε εύκολα να διαπιστώσετε μεταβαίνοντας σε κάποια κονσόλα με το συνδυασμό πλήκτρων [Alt+F1] και πληκτρολογώντας ένα whoami. Για επιστροφή στο περιβάλλον γραφικών πατήστε [Alt+F7], όπως ακριβώς γίνεται και σε κάθε desktop διανομή Linux.

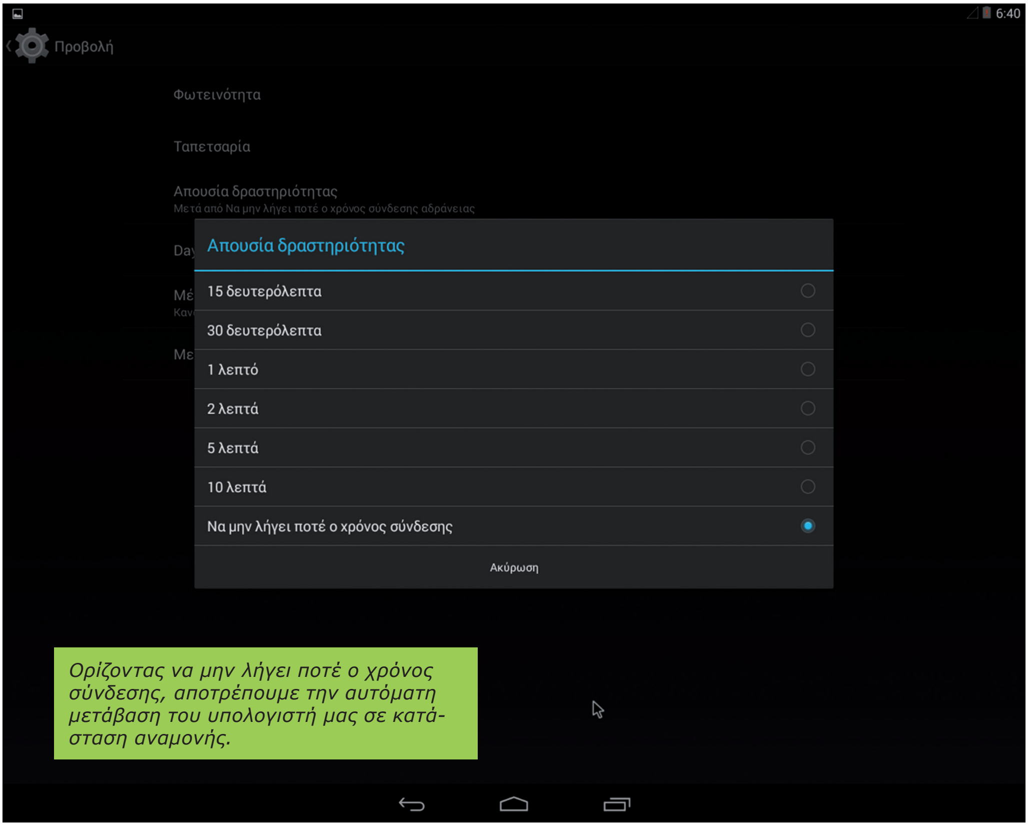




Σέρνοντας την άνω δεξιά γωνία με το ποντίκι, εμφανίζονται οι συντομεύσεις διάφορων λειτουργιών κι από εκεί θα μπούμε στο μενού ρυθμίσεων.

Θέματα συμβατότητας

Για έναν έλεγχο συμβατότητας, εγκαταστήσαμε το Android x86 τόσο σε laptop όσο και desktop σύστημα. Ο desktop υπολογιστής διέθετε μια σχετικά παλιά μητρική της MSI, η οποία έφερε πάνω της έναν διπύρηνο επεξεργαστή E2160 της Intel και 2GB μνήμης RAM, τύπου DDR2. Στη μητρική βρίσκονταν επίσης τοποθετημένη μια σχετικά παλιά PCI-Express κάρτα γραφικών με chip της ATI, ενώ το κύκλωμα του ήχου βρισκόταν ενσωματωμένο στο chipset της μητρικής. Όσον αφορά στα περιφερειακά, είχαμε συνδεδεμένη μια οθόνη 17" 4:3, καθώς κι από ένα πληκτρολόγιο και ποντίκι, συνδεδεμένα σε αντίστοιχες USB θύρες. Η δε σύνδεση με το δίκτυο γινόταν μέσω της ενσωματωμένης ενσύρματης κάρτας δικτύου. Το Android x86 συνεργάστηκε άψογα με το σύστημα και τα εγκατεστημένα περιφερειακά του. Η ανάλυση της επιφάνειας εργασίας ήταν στο μέγεθος της φυσικής ανάλυσης της οθόνης, ο ήχος αναπαραγόταν σωστά, όπως άλλωστε και το scrolling στα στοιχεία του UI, με τη χρήση της ροδέλας του ποντικού. Η σύνθεση του Laptop αποτελούνταν από έναν επεξεργαστή i5 της Intel στα 2.8GHz, 8GB μνήμης DDR3 και κύκλωμα γραφικών Intel HD. Φυσικά, το laptop είχε αναλογίες οθόνης 16:9, διαθέσιμη κάμερα, κάρτα ασύρματου δικτύου και bluetooth. Και σε αυτό το σύστημα, η συνεργασία με το Android x86 ήταν στα ίδια, υψηλά επίπεδα. Μοναδική ίσως παραφωνία η μη υποστήριξη του ενσωματωμένου κυκλώματος bluetooth. Περιττό νομίζω να πούμε ότι η απόκριση του περιβάλλοντος εργασίας του Android ήταν κατά πολύ μεγαλύτερη σε σχέση με ό,τι έχουμε δει μέχρι στιγμής από τις φορητές συσκευές που το υποστηρίζουν. Κι αυτό είναι πολύ φυσικό, καθώς η επεξεργαστική ισχύς των υλοποιήσεων της x86 πλατφόρμας είναι θηριώδης μπροστά σ' αυτές της ARM αρχιτεκτονικής, ακόμα και για την ίδια συχνότητα χρονισμού του επεξεργαστή.



Ορίζοντας να μην λήγει ποτέ ο χρόνος σύνδεσης, αποτρέπουμε την αυτόματη μετάβαση του υπολογιστή μας σε κατάσταση αναμονής.

Χρήση του Android στο PC

Το Android x86 στο PC μας συμπεριφέρεται σαν να βρίσκεται εγκατεστημένο σε ένα κινητό ή tablet. Έτσι, μη σας φανεί περίεργο αν πατώντας το κουμπί που βάζουμε σε λειτουργία τον υπολογιστή, κλειδώσει κι απενεργοποιηθεί η οθόνη και το σύστημα μπει σε κατάσταση αναμονής! Για επαναφορά σε λειτουργία πρέπει να ξαναπατήσουμε το διακόπτη λειτουργίας του PC. Φυσικά, αν η οθόνη έχει κλειδώσει θα πρέπει να την ξεκλειδώσουμε σέρνοντας το ποντίκι μας με πατημένο το αριστερό του κουμπί — όπως θα κάναμε και με την άκρη του δακτύλου μας σε μια οθόνη αφής.

Εάν όλα αυτά σας φαίνονται περίεργα, τότε πριν συνεχίσουμε την εργασία μας με το Android θα πρέπει να κάνουμε ορισμένες ρυθμίσεις. Κατ' αρχάς θα πρέπει να απενεργοποιήσουμε τον αυτόματο προσανατολισμό της οθόνης, ώστε να ελαχιστοποιήσουμε τις πιθανότητες να συστραφεί η επιφάνεια εργασίας σε γωνία διαφορετική προς το επίπεδο του γραφείου μας. Για να το κάνουμε αυτό, από τη δεξιά πάνω γωνία τραβάμε κι επεκτείνουμε την καρτέλα που εμφανίζει, σε μορφή κουμπιών, διάφορες βασικές συντομεύσεις. Επιλέγουμε κλείδωμα περιστροφής της οθόνης (βρίσκεται στο δεξί μέρος της δεύτερη σειράς των κουμπιών που εμφανίζονται). Αυτό φυσικά δεν θα αποτρέψει ορισμένες εφαρμογές που επιμένουν να εμφανίζουν το περιβάλλον τους οριζόντια.

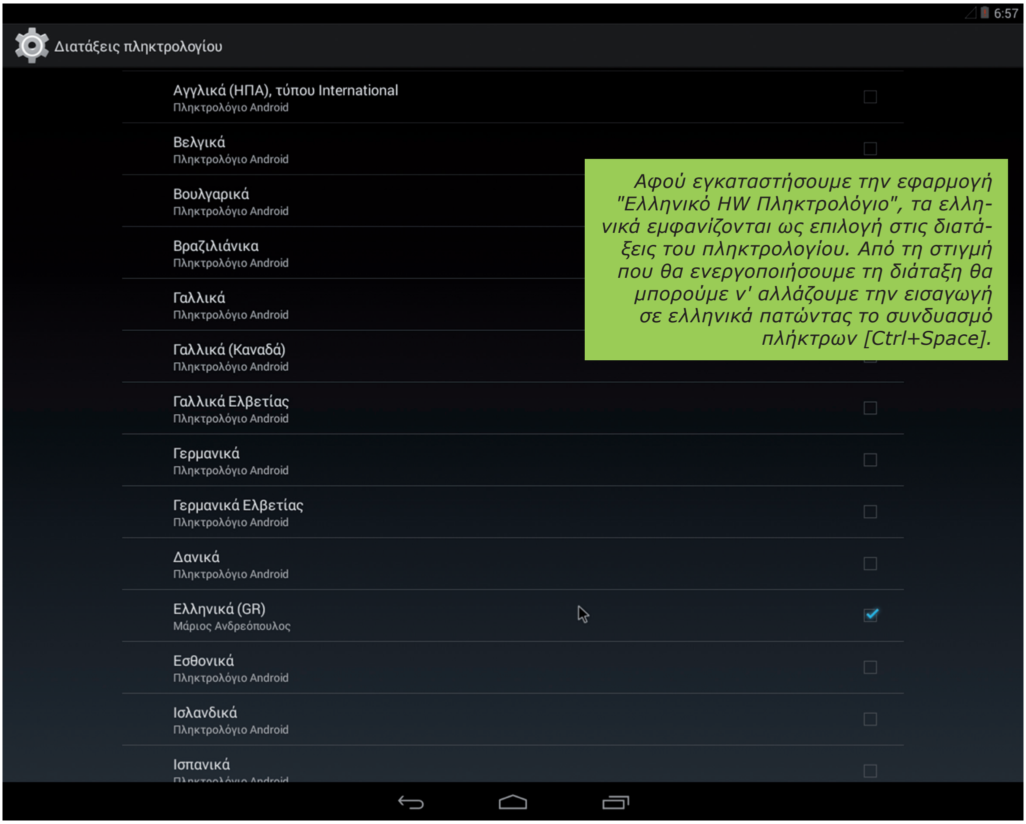


Με την ενεργοποίηση της εγκατάστασης εφαρμογών από άγνωστες πηγές, μπορούμε να τοποθετήσουμε στη συσκευή μας κι εφαρμογές που για διάφορους λόγους βρίσκονται εκτός του Google Play, αλλά κι αυτές που εμείς θα αναπτύξουμε.

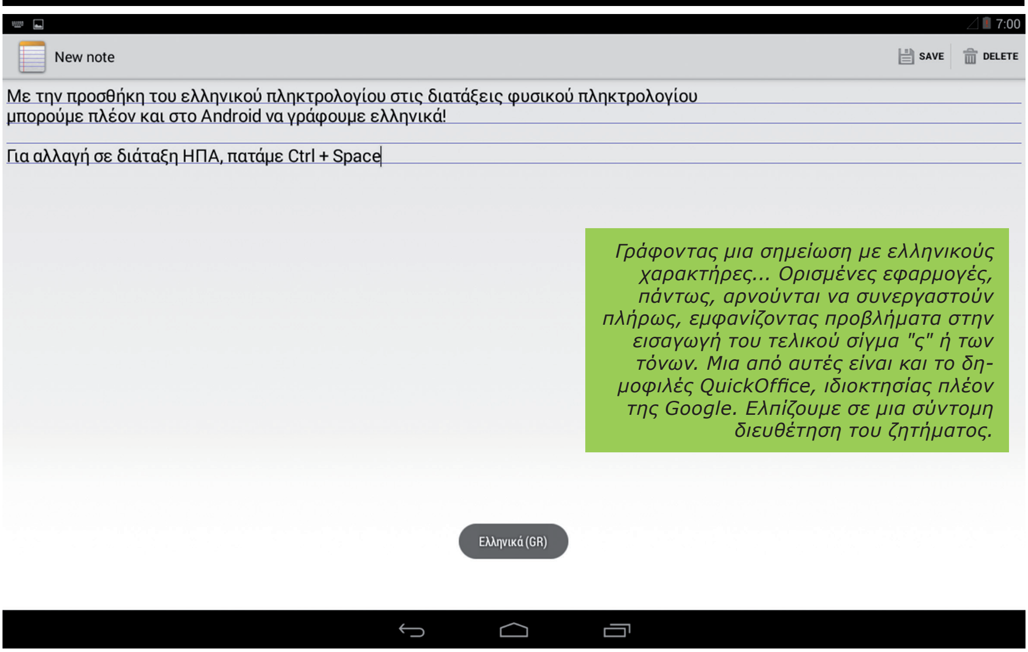
Από το ίδιο σημείο, κι επιλέγοντας το κουμπί των ρυθμίσεων, μεταβαίνουμε στο μενού ρυθμίσεων του Android. Από εκεί, μπορούμε να επιλέξουμε "Προβολή" και να θέσουμε την επιλογή "Απουσία δραστηριότητας" στο "Να μην λήγει ποτέ ο χρόνος σύνδεσης", έτσι ώστε να αποφύγουμε ενοχλητικές μεταβάσεις του PC σε κατάσταση αναμονής.

Πριν αποχωριστούμε το μενού ρυθμίσεων (Σ.τ.Ε. Όχι για πάντα :P), καλό θα ήταν να κάνουμε μια βόλτα κι από την επιλογή "Ασφάλεια" και στο "Κλείδωμα οθόνης" να βάλουμε το "Κανένα". Επίσης, αν μας ενδιαφέρει ο πειραματισμός και γενικώς το ψάξιμο του Android, ευρισκόμενοι στην ίδια καρτέλα, στην ενότητα "Διαχείριση συσκευής" μπορούμε να τσεκάρουμε την ρύθμιση "Άγνωστες πηγές", ώστε να μπορούμε να εγκαθιστούμε και εφαρμογές που δεν βρίσκονται εντός του Google Play. Παράδειγμα αποτελεί το δημοφιλές Adblock, όπως επίσης και τα δικά μας δημιουργήματα για την πλατφόρμα του Android.

Πέρα απ' αυτές τις ρυθμίσεις μας μένει να κάνουμε άλλη μια, ώστε να μπορούμε να απολαύσουμε πλήρως την εμπειρία χρήσης του Android λειτουργικού μας. Βλέπετε, από την έκδοση Jelly Bean κι έπειτα η εγγενής υποστήριξη του Android στη χρήση φυσικού πληκτρολογίου ωρίμασε αρκετά. Δυστυχώς, όμως, από τη λίστα των γλωσσών εισαγωγής, την οποία μπορείτε να βρείτε στο μενού ρυθμίσεων, στην επιλογή "Διατάξεις πληκτρολογίου" λείπουν τα ελληνικά. Ευτυχώς, υπάρχει μια έτοιμη λύση



Αφού εγκαταστήσουμε την εφαρμογή "Ελληνικό HW Πληκτρολόγιο", τα ελληνικά εμφανίζονται ως επιλογή στις διατάξεις του πληκτρολογίου. Από τη στιγμή που θα ενεργοποιήσουμε τη διάταξη θα μπορούμε ν' αλλάζουμε την εισαγωγή σε ελληνικά πατώντας το συνδυασμό πλήκτρων [Ctrl+Space].



Γράφοντας μια σημείωση με ελληνικούς χαρακτήρες... Ορισμένες εφαρμογές, πάντως, αρνούνται να συνεργαστούν πλήρως, εμφανίζοντας προβλήματα στην εισαγωγή του τελικού σίγμα "ς" ή των τόνων. Μια από αυτές είναι και το δημοφιλές QuickOffice, ιδιοκτησίας πλέον της Google. Ελπίζουμε σε μια σύντομη διευθέτηση του ζητήματος.

με όνομα "Ελληνικό HW Πληκτρολόγιο", την οποία μπορούμε να βρούμε στο κοντινότερό μας Google Play και να την εγκαταστήσουμε στο σύστημά μας. Μετά την εγκατάσταση της εφαρμογής, η ελληνική γλώσσα προστίθεται στο μενού των υποστηριζόμενων γλωσσών και μπορεί να ενεργοπηθεί μαζί με την αγγλική. Τώρα είμαστε σε θέση να γράφουμε ελληνικά και με το φυσικό πληκτρολόγιο. Για την αλλαγή της διάταξης της γλώσσας μην δοκιμάσετε το συνηθισμένο συνδυασμό [Alt+Shift], καθώς τίποτε δεν πρόκειται να γίνει. Ο συνδυασμός πλήκτρων στο Android για την αλλαγή της γλώσσας είναι [Ctrl+Space]. Μόλις τον πληκτρολογήσετε, μια μικρή, διακριτική ενημέρωση θα εμφανιστεί στο κάτω μέρος της οθόνης, πληροφορώντας σας για τη γλώσσα εισαγωγής που μόλις ενεργοποιήσατε.

Η προετοιμασία για τον πειραματισμό του Android συστήματός που τρέχει στο PC μας, μόλις ολοκληρώθηκε. Αφήστε τη φαντασία σας ελεύθερη, αξιοποιώντας με ευρηματικούς τρόπους το νέο σας λειτουργικό σύστημα!

Σε διάφορα σημεία του κειμένου, τόσο στο πρώτο όσο και στο δεύτερο μέρους του μίνι αφιερώματός μας, περιγράφουμε τις διάφορες εκδόσεις του Android είτε αριθμητικά είτε με τις κωδικές ονομασίες της εκάστοτε έκδοσης. Κατά παράδοση, από το 2009 για κάθε "μεγάλη" έκδοση του Android η Google επιλέγει ένα κωδικό όνομα κάποιου γλυκίσματος. Η λίστα των ονομασιών περιλαμβάνει:

- **Cupcake (1.5)**
- **Doughnut (1.6)**
- **Eclair (2.0–2.1)**
- **Froyo (2.2–2.2.3)**
- **Gingerbread (2.3–2.3.7)**
- **Honeycomb (3.0–3.2.6)**
- **Ice Cream Sandwich (4.0–4.0.4)**
- **Jelly Bean (4.1–4.3.1)**
- **KitKat (4.4–4.4.4)**

Για τις πρότερες εκδόσεις του Android, υπάρχουν διάφορες απόψεις. Το μόνο σίγουρο είναι πως δεν υπάρχει "επίσημη" ονοματολογία. Σύμφωνα πάντως με τον Jean-Baptiste Quéru, μηχανικό του Android, η έκδοση 1.0 δεν έφερε κάποιο κωδικό όνομα, εν αντιθέσει με την 1.1 η οποία εσωτερικά ονομάστηκε Petit Four κι από εκεί κι έπειτα δόθηκε το έναυσμα για τις "γλυκειές" ονοματοδοσίες, αλλά με αλφαβητική σειρά. Τέλος, για την επόμενη έκδοση, την 5, όπου σύντομα αναμένουμε την ανακοίνωσή της, κυκλοφορούν φήμες περί ονομασίας της ως Lime Pie ή Lollipop. Φυσικά, δεν έχουμε παρά να περιμένουμε την επίσημη ανακοίνωση, ώστε να δούμε αν επιβεβαιώνονται οι φήμες :)

Οι "γλυκειές" ονομασίες των διάφορων εκδόσεων του Android, μαζί με τα λογότυπά τους.



Skill: Intermediate
Tags: Entropy Pools, Randomness, SoX, Python

Let's generate some noise



by Spir@l Evolut10n

Δεν πέτυχε, έτυχε!

Κι αυτό όχι μόνο δεν είναι κακό, αλλά σε πολλές περιπτώσεις είναι κι απόλυτα επιθυμητό. Μόνο που, όπως η δική μας τύχη, έτσι κι αυτή των υπολογιστών συχνά εξαντλείται και τότε συμβαίνουν δυσάρεστα πράγματα. Ευτυχώς πάντως που στην περίπτωση των υπολογιστών μπορούμε να λάβουμε τα μέτρα μας ώστε να μην ξεμένουμε ποτέ από τύχη.

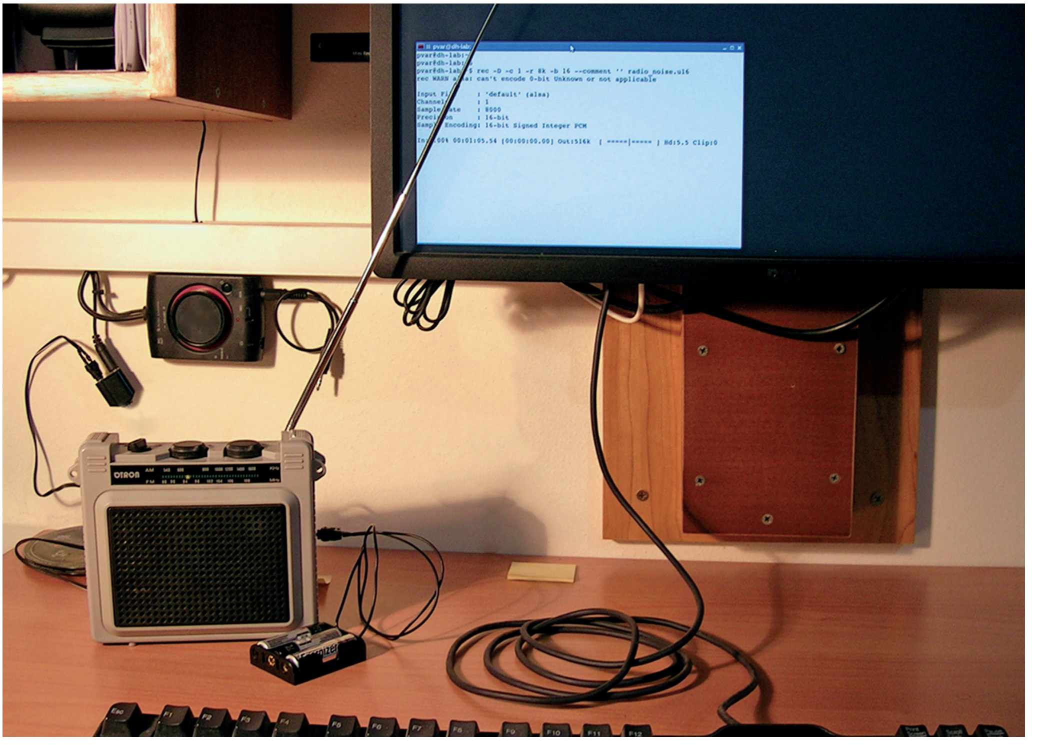
Στο τεύχος 031 του deltaHacker ο αγαπητός συνεργάτης George of the Jungle αναφέρθηκε στη σπουδαιότητα που έχουν οι τυχαίοι αριθμοί για την κρυπτογραφία[1]. Η τυχαιότητα όμως αξιοποιείται και σε πολλά άλλα πεδία της Πληροφορικής. Σκεφτείτε για παράδειγμα το μηχανισμό ASLR (Address Space Layout Randomization), που ενσωματώνουν όλα τα σύγχρονα λειτουργικά συστήματα για να δυσκολεύουν τη ζωή των reversers. Ο εν λόγω μηχανισμός τοποθετεί στη στοίβα τον κώδικα των βιβλιοθηκών αλλά και τον ίδιο τον κώδικα του εκάστοτε προγράμματος σε τυχαίες διευθύνσεις της μνήμης. Έτσι, όταν εφαρμόζει κανείς την τεχνική stack overflow για την παράνομη εκτέλεση ενός τμήματος κώδικα, είναι αδύνατο να επαναλάβει τη διαδικασία χρησιμοποιώντας τις ίδιες διευθύνσεις μνήμης. Κατ' επέκταση, η τυποποίηση κι ο αυτοματισμός της επίθεσης είναι από δύσκολος έως αδύνατος. Ένα άλλο παράδειγμα χρήσης των τυχαίων αριθμών αποτελεί το λεγόμενο disk wiping: Πρόκειται για τη μόνιμη κι ασφαλή εκκαθάριση ενός δίσκου, που επιτυγχάνεται με την εγγραφή τυχαίων δεδομένων στο σύνολο της έκτασής του. Η απαίτηση για τυχαίους αριθμούς αποτελεί και το λόγο για τον οποίο το disk wiping καθυστερεί *πολύ* περισσότερο από μια απλή διαγραφή. Νομίζουμε όμως ότι δεν χρειάζονται περισσότερα παραδείγματα. Είναι βέβαιο ότι γνωρίζετε κι εσείς προγράμματα που απαιτούν άφθονους τυχαίους αριθμούς και τα οποία δεν θα θέλατε να παρουσιάζουν προβλέψιμη συμπεριφορά, αλλά ούτε αδικαιολόγητες καθυστερήσεις. Αν όλα τα προηγούμενα φαντάζουν πολύ θεωρητικά και σας αφήνουν αδιάφορους, θα πρέπει να το ξανασκεφτείτε. Ακόμα κι αν εσείς οι ίδιοι δεν πρόκειται ν' ασχοληθείτε ποτέ με την κρυπτογραφία, είναι πολύ πιθανό να ασχολείται ήδη το σύστημά σας! Για παράδειγμα, ένας web server που δέχεται κρυπτογραφημένες συνδέσεις HTTPS, παράγει κλειδιά κρυπτογράφησης διαρκώς και για κάθε νέο επισκέπτη. Το ίδιο ισχύει και για έναν mail server, που δέχεται και στέλνει την αλληλογραφία μέσω κρυπτογραφημένων καναλιών επικοινωνίας.

Σ' αυτό το άρθρο θα εξετάσουμε δύο μεθόδους για την παραγωγή τυχαίων αριθμών, με τη βοήθεια του hardware. Μην πάει ο νους σας σε εξωτικές ή/και πανάκριβες συσκευές. Αντίθετα, θα στηριχτούμε σε εξοπλισμό που είναι βέβαιο ότι διαθέτουν όλοι. Επομένως, είτε έχετε ανάγκη από μεγάλα αποθέματα τυχαίων αριθμών είτε όχι, το σίγουρο είναι ότι δεν έχετε καμία *καλή* δικαιολογία ώστε να μην πειραματιστείτε με όσα θα δείξουμε ;) Η εργασία μας θα πραγματοποιηθεί σε περιβάλλον Linux, ενώ θα επιστρατεύσουμε και την αγαπημένη μας Python. (Σ.τ.Ε. Ποιος είπε ότι το καλοκαίρι πρέπει να υποχωρεί το geek factor;)

Δεξαμενές εντροπίας

Στα συστήματα Linux ο πυρήνας ενσωματώνει έναν μηχανισμό που φροντίζει για την παραγωγή τυχαίων αριθμών. Ο εν λόγω μηχανισμός αποτελεί μια γεννήτρια τυχαίων αριθμών (Random Number Generator, RNG), η οποία τροφοδοτείται με "δείγματα" από διάφορα υποσυστήματα του λειτουργικού, των οποίων η συμπεριφορά θεωρείται απρόβλεπτη. Σκεφτείτε για παράδειγμα τη δραστηριότητα των δίσκων, τη δικτυακή κίνηση, τα πατήματα πλήκτρων, την κίνηση του ποντικιού κ.λπ. Όπως αντιλαμβάνεστε, ο ρυθμός με τον οποίο συγκεντρώνονται τα δείγματα καθορίζεται από τη γενικότερη δραστηριότητα του συστήματος και είναι απρόβλεπτος. Γι’ αυτό το λόγο, τα τυχαία δείγματα αποθηκεύονται προσωρινά σ' ένα "δοχείο" που ονομάζεται δεξαμενή εντροπίας (entropy pool). Παρεμπιπτόντως, να σημειώσουμε ότι η εντροπία στην Πληροφορική δεν έχει την ίδια έννοια με την εντροπία στη Φυσική. Στην Πλη-

1. Δείτε τα άρθρα στις σελίδες 24 και 62, στο τεύχος 031.



Το παλιό ραδιοφωνάκι που ξετρυπώσαμε απόκτησε νέα ζωή! Συνδέοντας την έξοδο των ακουστικών στην είσοδο της κάρτας ήχου του συστήματος, ηχογραφήσαμε ραδιοφωνικά παράσιτα για αρκετές ώρες και δημιουργήσαμε μια πλούσια δεξαμενή εντροπίας.

ροφορική, η εντροπία αποτελεί μέτρο της αβεβαιότητας και όχι της αταξίας, όπως συμβαίνει στη Φυσική. Με απλά λόγια, η έννοια της εντροπίας στην Πληροφορική συγγενεύει με αυτήν της τυχαιότητας. Ας επιστρέψουμε στο θέμα όμως.

Κάθε φορά που η γεννήτρια τυχαίων αριθμών εξαντλεί τη δεξαμενή εντροπίας, τα δεδομένα της δεξαμενής θεωρούνται ακατάλληλα για περαιτέρω χρήση. Έτσι, η γεννήτρια παύει προσωρινά τη λειτουργία της και περιμένει έως ότου η δεξαμενή αποκτήσει το κατάλληλο πλήθος νέων δειγμάτων. Οι τυχαίοι αριθμοί που παράγονται μ' αυτόν τον τρόπο θεωρούνται ασφαλείς για χρήση σε εφαρμογές κρυπτογραφίας και διανέμονται στο σύστημα μέσω ενός device file, που δεν είναι άλλο από το /dev/random. Οι παύσεις που αναφέραμε, όμως, έχουν σαν συνέπεια τη χαμηλή απόδοση της γεννήτριας: Τον περιορισμένο ρυθμό παραγωγής τυχαίων αριθμών. Εξάλλου, όταν τα αποθέματα της δεξαμενής εντροπίας εξαντλούνται, το σύστημα μπλοκάρει την ανάγνωση του /dev/random, έως ότου η δεξαμενή αποκτήσει νέα δείγματα.

Εξαιτίας των παραπάνω κι επειδή υπάρχουν εφαρμογές που απαιτούν πολλούς τυχαίους αριθμούς, τα συστήματα Linux προσφέρουν και μια εναλλακτική πηγή τυχαίων αριθμών. Αναφερόμαστε στο device file /dev/urandom (από το "unlimited random"), που είναι ανεξάντλητο. Αναρωτιέστε πώς επιτυγχάνεται αυτό; Πολύ απλά, κάθε φορά που η δεξαμενή εντροπίας εξαντλείται, η γεννήτρια για το /dev/urandom δεν περιμένει την "αναγόμωση" της δεξαμενής κι άρα η λειτουργία της δεν σταματά

ποτέ. Η συγκεκριμένη γεννήτρια ανακυκλώνει τα υπάρχοντα περιεχόμενα της δεξαμενής και παράγει ψευδοτυχαίους αριθμούς υποδεέστερης "ποιότητας". Αυτοί οι αριθμοί θεωρούνται "αρκετά τυχαίοι" για τις περισσότερες εφαρμογές, αλλά αντιμετωπίζονται σαν προβλέψιμοι και επισφαλείς για χρήση στην κρυπτογραφία.

Στις πρόσφατες εκδόσεις του πυρήνα του Linux, το μέγεθος της δεξαμενής εντροπίας είναι "καρφωμένο" στον κώδικα (hardcoded) και έχει οριστεί στα 4096 bits (ή 512 bytes). Αν έχετε την περιέργεια, ο σχετικός ορισμός βρίσκεται στη γραμμή 275 του αρχείου /path_to_kernel_source/drivers/char/random.c:

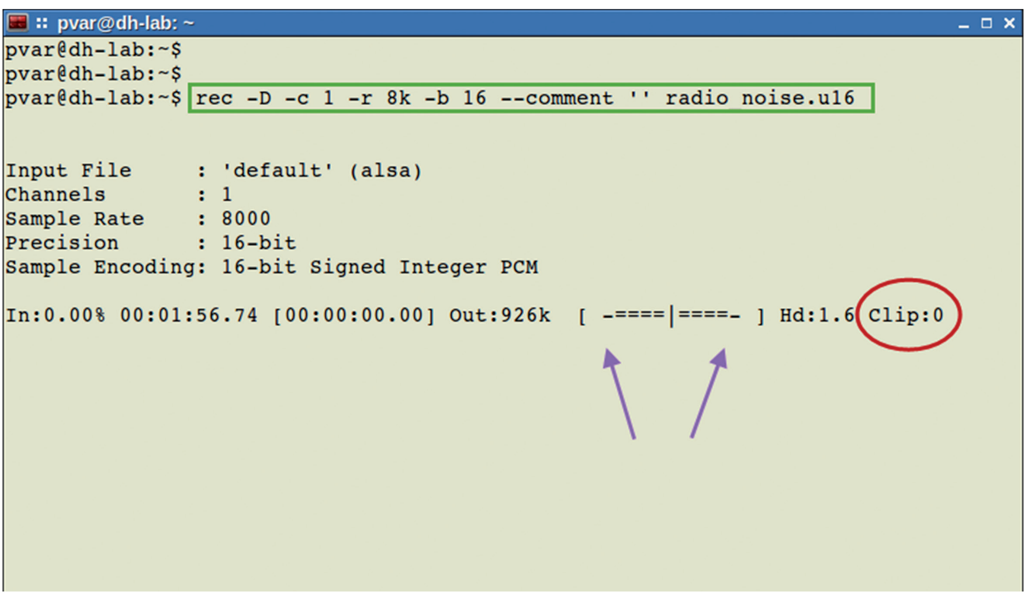
```
/*
* Configuration information
*/
#define INPUT_POOL_WORDS 128
#define OUTPUT_POOL_WORDS 32
```

Η χωρητικότητα της δεξαμενής εντροπίας σε bits προκύπτει από το γινόμενο των INPUT_POOL_WORDS με το OUTPUT_POOL_WORDS. Πάντως, για τους ακόμα πιο περίεργους, αν θέλετε να τσεκάρετε το μέγεθος της δεξαμενής στο δικό σας σύστημα, μπορείτε να εκτελέσετε στη γραμμή εντολών αυτό

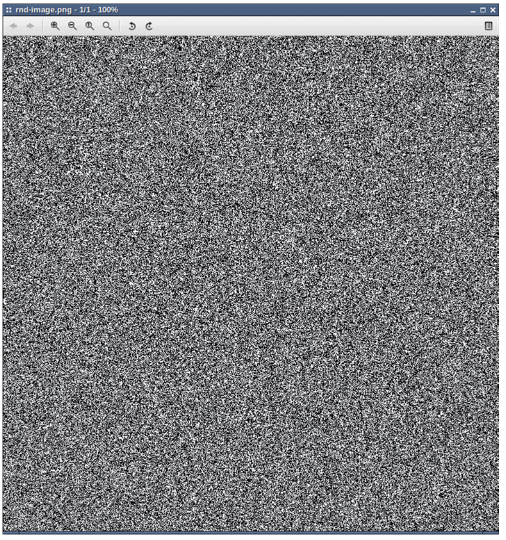
```
cat /proc/sys/kernel/random/poolsize
```

ή αυτό

```
sysctl kernel.random.poolsize
```



Η ηχογράφηση πραγματοποιήθηκε με το προγραμματάκι rec. Πριν αφήσουμε το σύστημα να καταγράψει τα ραδιοφωνικά παράσιτα για καμπόσες ώρες, τσεκάραμε ότι η ένταση του ήχου ήταν σχετικά χαμηλή και δεν παραμορφωνόταν. Το rec φροντίζει να ενημερώνει το χρήστη για την ένταση του "εισερχόμενου" σήματος με μερικούς χαρακτήρες που σχηματίζουν ένα VU meter, ενώ ταυτόχρονα ενημερώνει και για το αν προκαλείται clipping.



Αφού επεξεργαστήκαμε τα δείγματα θορύβου και δημιουργήσαμε τους τυχαίους αριθμούς, προχωρήσαμε σε έναν πρώτο (και μάλλον απλοϊκό) έλεγχο: Οπτικοποιήσαμε ένα μέρος των τυχαίων αριθμών και ελέγξαμε για την ύπαρξη επαναλαμβανόμενων μοτίβων. Όπως βλέπετε, οι αριθμοί μας μοιάζουν αρκετά τυχαίοι :P

Σημειώστε ότι τα μεγέθη που εμφανίζονται με την εκτέλεση των παραπάνω αναφέρονται σε bits. Επιπρόσθετα, αν θέλετε να τσεκάρετε το διαθέσιμο απόθεμα εντροπίας (πάλι σε bits), αρκεί να δώσετε αυτό:

```
cat /proc/sys/kernel/random/entropy_
avail
```

Ωραία όλα αυτά, θα πείτε, αλλά γιατί πρέπει να τα έχουμε υπόψη μας; Όπως αναφέραμε, οι τυχαίοι αριθμοί που θεωρούνται ασφαλείς για χρήση στην κρυπτογραφία προκύπτουν από την επεξεργασία των απρόβλεπτων μεγεθών που συλλέγει ο πυρήνας. Σκεφτείτε όμως την περίπτωση ενός headless server, με τον οποίο αλληλεπιδρούμε ελάχιστα. Τα τυχαία μεγέθη που θα μπορούσε να συλλέξει ο πυρήνας από το πληκτρολόγιο ή το ποντίκι απουσιάζουν. Ακόμα χειρότερα, αν μιλάμε για κάποιο VPS, η δικτυακή κίνηση και η πρόσβαση στους δίσκους ελέγχεται και σε κάποιο βαθμό ρυθμίζεται/επηρεάζεται από τον εκάστοτε hypervisor. Επομένως, τα σχετικά μεγέθη ενδέχεται να παρουσιάζουν αυξημένη κανονικότητα και, τέλος πάντων, να είναι "λιγότερο τυχαία". Τα παραπάνω συνεπάγονται τον αργό ρυθμό ανανέωσης της δεξαμενής εντροπίας και, κατ' επέκταση, τα συχνά μπλοκαρίσματα στην ανάγνωση του /dev/random. (Σ.τ.Ε. Τις προάλλες ήθελα να δημιουργήσω σε κάποιο VPS ένα αρχείο με τυχαία δεδομένα από το /dev/random, κι αναρωτιόμουν γιατί αποτύγχανα :/) Αυτά τα μπλοκαρίσματα, για ένα VPS που λειτουργεί ως web server και υποστηρίζει συνδέσεις HTTPS, είναι πιθανό να εκφραστούν κι ως καθυστερήσεις στην εξυπηρέτηση των επισκεπτών.

Από τα παραπάνω, συμπεραίνουμε ότι αν επιθυμούμε *πολλούς* και πραγματικά τυχαίους αριθμούς στο Linux, έχουμε δύο επιλογές: Είτε να φροντίσουμε μόνοι μας για την παραγωγή αξιόπιστων τυχαίων αριθμών είτε να εξασφαλίσουμε την ταχύτερη ανανέωση της δεξαμενής εντροπίας, με μεγέθη που μεταβάλλονται απρόβλεπτα. Στη συνέχεια του άρθρου θα εξετάσουμε μια λύση για κάθε σενάριο.

Μόνο στο Linux!

Δεξαμενή εντροπίας διατηρούν όλα τα σύγχρονα λειτουργικά συστήματα και, φυσικά, όλα τα συστήματα της οικογένειας του Unix. Μάλιστα, για την τροφοδότηση των δεξαμενών χρησιμοποιούνται λίγο-πολύ οι ίδιες πηγές: Δραστηριότητα αποθηκευτικών μέσων, δικτυακή κίνηση, πληκτρολόγιο, ποντίκι κ.λπ. Ωστόσο, στην περίπτωση των FreeBSD, MAC OS X και iOS, ο πυρήνας επεξεργάζεται τα δεδομένα της δεξαμενής με έναν αλγόριθμο παραγωγής ψευδοτυχαίων αριθμών που ονομάζεται Yarrow. Ο συγκεκριμένος αλγόριθμος μπορεί να "ανακυκλώνει"

τα περιεχόμενα της δεξαμενής εντροπίας και να λειτουργεί ακατάπαυστα, ενώ οι αριθμοί που παράγει θεωρούνται αξιόπιστοι για χρήση σε εφαρμογές κρυπτογραφίας. Σε αυτά τα συστήματα, λοιπόν, το /dev/random και το /dev/urandom παρουσιάζουν πανομοιότυπη συμπεριφορά. Σε ορισμένες περιπτώσεις, μάλιστα, το δεύτερο αποτελεί σύνδεσμο (link) προς το πρώτο. Τα υπόλοιπα συστήματα Unix ενσωματώνουν παραλλαγές του Yarrow ή και εντελώς διαφορετικούς αλγόριθμους. Επομένως, οι διαφορές που αναφέραμε μεταξύ random και urandom ισχύουν αποκλειστικά στα συστήματα Linux.

Υπάρχουν παντού

Ο θόρυβος καταφέρνει να εισβάλλει σε κάθε χώρο αλλά και σε κάθε φυσική διαδικασία. Τα μηχανικά κύματα που αντιλαμβανόμαστε ως ήχο, π.χ., περιέχουν πάντοτε θόρυβο. Κάποιες φορές είναι έντονος κι αντιληπτός και κάποιες άλλες ανεπαίσθητος, αλλά πάντα υπαρκτός. Το ίδιο ισχύει και για τον "θερμικό θόρυβο" που προκαλεί τις τυχαίες ταλαντώσεις στους φορείς του ηλεκτρικού φορτίου σε έναν αγωγό. Σε αυτό το είδος θορύβου οφείλεται κατά ένα μέρος και ο ηλεκτρικός θόρυβος, που εμφανίζεται στα ηλεκτρικά σήματα. Θόρυβος παρατηρείται ακόμα και στις γονιδιακές λειτουργίες, όπου εκφράζεται με τις απρόσμενες κυτταρικές μεταλλάξεις. Αναρωτιέστε τι μας έπιασε και μιλάμε για το θόρυβο; Μιλώντας αφηρημένα, θα μπορούσαμε να πούμε ότι ο θόρυβος αποτελεί ένα σύμπλεγμα απρόβλεπτων διακυμάνσεων, που δεν παρουσιάζει καμία περιοδικότητα. Με άλλα λόγια, ο θόρυβος αποτελεί την ιδανική πηγή τυχαιότητας!

Οι τυχαίοι αριθμοί, λοιπόν, βρίσκονται σε κάθε φυσική διαδικασία και για την παραγωγή τους θα μπορούσαμε να εκμεταλλευτούμε οποιοδήποτε φαινόμενο. Δείτε μερικές πηγές που χρησιμοποιούνται συχνά:

- Ηλεκτρικός θόρυβος από το φαινόμενο της "χιονοστιβάδας" σε ανάστροφα πολωμένη επαφή PN
- Ηλεκτρικός θόρυβος σε ταλαντωτή δακτυλίου (ring oscillator)
- Θόρυβος από μικρόφωνο στραμμένο στον άνεμο
- Λυχνία Γκάιγκερ (Geiger tube) για την παρατήρηση των τυχαίων διασπάσεων των ατόμων ενός ραδιενεργού υλικού
- Θόρυβος υποβάθρου από ραδιόφωνο που *δεν* έχει συντονιστεί σε σταθμό

Μπορεί κάποιες από τις παραπάνω μεθόδους να μοιάζουν απλοϊκές κι ως εκ τούτου αναξιόπιστες, αλλά σας διαβεβαιώνουμε ότι δεν είναι. Για παράδειγμα, η υπηρεσία www.random.org που είναι αρκετά παλιά και διάσημη στο χώρο, στηρίζεται στη μέθοδο με το ραδιόφωνο! Ε, λοιπόν, την ίδια μέθοδο θα υλοποιήσουμε κι εμείς. Θα δημιουργήσουμε ένα σύνολο τυχαίων αριθμών, με "πρώτη ύλη" τα παράσιτα του ραδιοφώνου.

Αποσυντονιστείτε!

Προφανώς, για την εργασία μας θα χρειαστούμε ένα απλό ραδιοφωνάκι. Το πώς ακριβώς θα πάρουμε και θα αξιοποιήσουμε τον θόρυβο, θα το δούμε αργότερα. Για την ώρα θα ασχοληθούμε με την απαιτούμενη ρύθμιση του δέκτη. Κατ' αρχάς δεν περιελάμβανε, σημειώστε ότι πρέπει να βρούμε ένα παλιό ραδιόφωνο, με όσο το

```
pvar@dh-lab:~$
pvar@dh-lab:~$ ent rnd.bin
Entropy = 7.999990 bits per byte.

Optimum compression would reduce the size
of this 19445956 byte file by 0 percent.

Chi square distribution for 19445956 samples is 259.27, and randomly
would exceed this value 50.00 percent of the times.

Arithmetic mean value of data bytes is 127.5428 (127.5 = random).
Monte Carlo value for Pi is 3.141230833 (error 0.01 percent).
Serial correlation coefficient is 0.000023 (totally uncorrelated = 0.0).
pvar@dh-lab:~$
pvar@dh-lab:~$
```

Μεταξύ άλλων, το προγραμματάκι ent επιστρατεύει τη στατιστική μέθοδο Monte Carlo, για τον πειραματικό υπολογισμό της τιμής του π. Κρίνοντας από την μικρή απόκλιση στο σφάλμα υπολογισμού του π, συμπεραίνουμε ότι οι αριθμοί που δώσαμε στο ent είναι "εντελώς τυχαίοι".

δυνατόν λιγότερους αυτοματισμούς. Πολλά μοντέρνα ραδιόφωνα, βλέπετε, εντοπίζουν τους σταθμούς και συντονίζονται αυτόματα. Κάποια άλλα, ενσωματώνουν φίλτρα που περιορίζουν το θόρυβο (noise reduction). Περιττό να πούμε ότι οι λειτουργίες αυτού του είδους είναι εντελώς ανεπιθύμητες για την εργασία μας. Επιπρόσθετα, το ραδιοφωνάκι μας πρέπει να έχει την επιλογή για λήψη

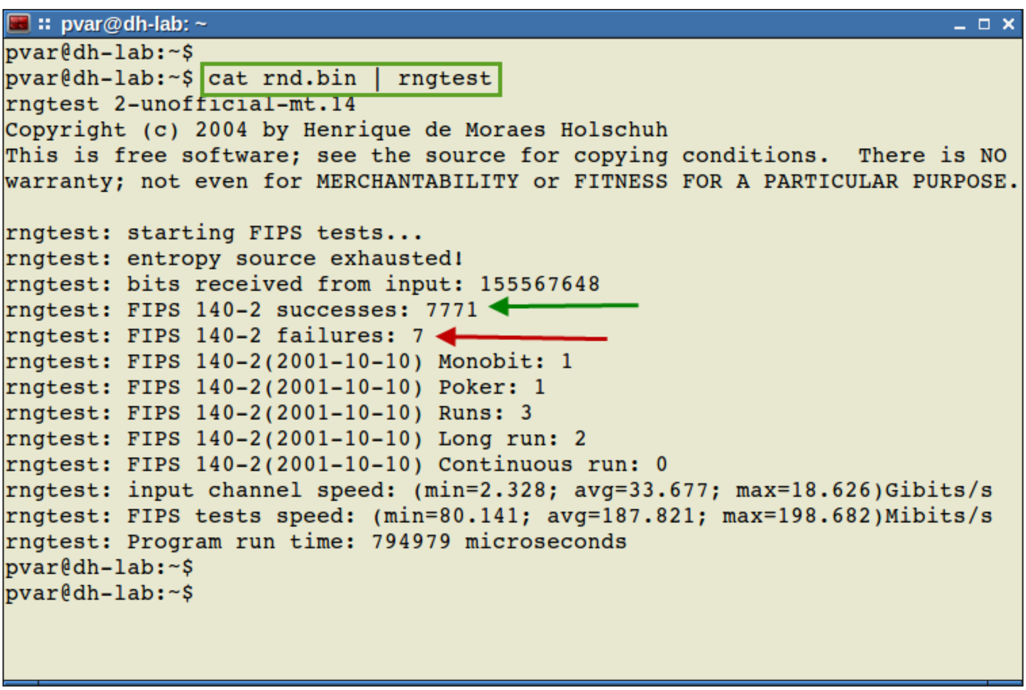
σημάτων AM. Αυτή η ανάγκη προκύπτει από τον τρόπο μετάδοσης της πληροφορίας σε κάθε είδος διαμόρφωσης. Στην περίπτωση των σημάτων FM, η πληροφορία μεταφέρεται σε μια *περιοχή* συχνοτήτων. Αυτό σημαίνει ότι ακόμα κι όταν δεν έχουμε συντονιστεί στην κύρια συχνότητα εκπομπής ενός σταθμού, είναι πιθανό να λαμβάνουμε ένα μέρος του σήματος του. Αντίθετα, στην περίπτωση των σημάτων AM, η πληροφορία μεταφέρεται σε μία *συχνότητα* και μόνο (τουλάχιστον θεωρητικά). Έτσι, όταν δεν έχουμε συντονιστεί στη συχνότητα εκπομπής ενός σταθμού, η λήψη μας επηρεάζεται ελάχιστα έως καθόλου από το αντίστοιχο σήμα.

Ανακεφαλαιώνοντας, χρειαζόμαστε ένα ραδιοφωνάκι χωρίς φίλτρα βελτίωσης του ήχου, το οποίο θα ρυθμίσουμε για λήψη σημάτων AM. Ο δέκτης του πρέπει να συντονιστεί σε κάποια συχνότητα, μακριά από οποιονδήποτε σταθμό. Αν ακούμε σε όλες τις συχνότητες κάποιον βόμβο —ή κάποιο σήμα που παρουσιάζει περιοδικότητα— δεν υπάρχει λόγος ανησυχίας. Αρκεί να συντονιστούμε σε κάποια συχνότητα, στην οποία τα παράσιτα ακούγονται δυνατότερα από κάθε άλλο σήμα.

Διασύνδεση και εγγραφή

Αφού ετοιμάσουμε το ραδιοφωνάκι, μπορούμε να προχωρήσουμε στην καταγραφή του θορύβου. Κάπου εδώ θα μπει στο παιχνίδι και ο υπολογιστής. Για τη διασύνδεση των δύο θα χρειαστούμε ένα απλό καλώδιο "jack to jack": Το ένα άκρο θα συνδεθεί στην έξοδο των ακουστικών του ραδιοφώνου και το άλλο στην είσοδο line-in της κάρτας ήχου του υπολογιστή.

Σειρά έχει η εγκατάσταση του εργαλείου για την καταγραφή του θορύβου. Γι' αυτό το σκοπό επιλέξαμε το προγραμματάκι rec, που αποτελεί τμήμα του πακέτου SoX (http://sox.sourceforge.net). Το rec επιτρέπει στο χρήστη να καθορίσει το ρυθμό δειγματοληψίας, το πλήθος των bits ανά δείγμα, το φορμά του αρχείου εγγραφής κ.ά. Κατ' αρχάς, πρέπει να έχουμε υπόψη ότι η διαδικασία ψηφιοποίησης αλλοιώνει πάντα το αρχικό περιεχόμενο του ήχου — έστω κι ανεπαίσθητα. Το ίδιο θα συμβεί και με τον πολύτιμο θόρυβο, από τον οποίο επιδιώκουμε να εξάγουμε τους τυχαίους αριθμούς. Αυτό δεν μπορούμε να το αποτρέψουμε, αλλά μπορούμε να εξαλείψουμε κάθε άλλο παράγοντα αλλοίωσης. Έτσι, θα προτιμήσουμε την καταγραφή σε αρχείο RAW, το οποίο δεν προβλέπει συμπίεση και δεν επηρεάζει καθόλου το σήμα. Εξάλλου, τα αρχεία αυτού του είδους είναι ιδανικά για την εργασία μας και για έναν πρόσθετο λόγο: Η εσωτερική τους δομή είναι εξαιρετικά απλή και θα διευκολύνει σε μεγάλο βαθμό τη μετέπειτα επεξεργασία των δειγμάτων. Εφόσον μας ενδιαφέρει η τυχαιότητα που κρύβεται μέσα στο θόρυβο, η καταγραφή δύο ή περισσοτέρων καναλιών δεν θα μας ωφελήσει σε τίποτα, καθώς τα αντίστοιχα σήματα θα είναι πλήρως συσχετισμένα μεταξύ τους! Για τη συχνότητα δειγματοληψίας θέλουμε μια "μεσαία" επιλογή, όπως τα 16KHz. Αν καταγράψουμε το θόρυβο με πολύ χαμηλότερο ρυθμό δειγματοληψίας, θα χάσουμε ένα σημαντικό τμήμα του φάσματός του. Από την άλλη, τα περισσότερα παλιά ραδιόφωνα διαθέτουν ενισχυτές που απορρίπτουν τις υψηλές συχνότητες ή, πιο απλά, αδυνατούν να τις διαχειριστούν. Επομένως, ο θόρυβος που θα πάρουμε από ένα παλιό ραδιόφωνο θα είναι απαλλαγμένος από τις


```
:: pvar@dh-lab: ~
pvar@dh-lab:~$
pvar@dh-lab:~$ cat rnd.bin | rngtest
rngtest 2-unofficial-mt.14
Copyright (c) 2004 by Henrique de Moraes Holschuh
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

rngtest: starting FIPS tests...
rngtest: entropy source exhausted!
rngtest: bits received from input: 155567648
rngtest: FIPS 140-2 successes: 7771
rngtest: FIPS 140-2 failures: 7
rngtest: FIPS 140-2(2001-10-10) Monobit: 1
rngtest: FIPS 140-2(2001-10-10) Poker: 1
rngtest: FIPS 140-2(2001-10-10) Runs: 3
rngtest: FIPS 140-2(2001-10-10) Long run: 2
rngtest: FIPS 140-2(2001-10-10) Continuous run: 0
rngtest: input channel speed: (min=2.328; avg=33.677; max=18.626)Gibits/s
rngtest: FIPS tests speed: (min=80.141; avg=187.821; max=198.682)Mibits/s
rngtest: Program run time: 794979 microseconds
pvar@dh-lab:~$
pvar@dh-lab:~$
```

Το rng-test πραγματοποιεί τους στατιστικούς ελέγχους που περιγράφονται στο πρότυπο FIPS 140-2. Οι εν λόγω έλεγχοι χρησιμοποιούνται για την πιστοποίηση συστημάτων κρυπτογράφησης, ως κατάλληλα για χρήση σε δημόσιες υπηρεσίες που χειρίζονται ευαίσθητα προσωπικά δεδομένα. Όπως βλέπετε, οι τυχαίοι αριθμοί που δημιουργήσαμε περνούν τους σχετικούς ελέγχους με τεράστιο ποσοστό επιτυχίας.

```
:: pvar@dh-lab: ~
#=============================================================================#
#            dieharder version 3.31.1 Copyright 2003 Robert G. Brown          #
#=============================================================================#
   rng_name    |           filename             |rands/second|
 file_input_raw|                         rnd.bin|  6.08e+07  |
#=============================================================================#
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
#=============================================================================#
# The file file_input_raw was rewound 2 times
   diehard_birthdays|   0|       100|     100|0.31716760|  PASSED
# The file file_input_raw was rewound 23 times
      diehard_operm5|   0|   1000000|     100|0.03713086|  PASSED
# The file file_input_raw was rewound 49 times
  diehard_rank_32x32|   0|     40000|     100|0.68630659|  PASSED
# The file file_input_raw was rewound 62 times
    diehard_rank_6x8|   0|    100000|     100|0.53230737|  PASSED
# The file file_input_raw was rewound 67 times
   diehard_bitstream|   0|   2097152|     100|0.87197412|  PASSED
# The file file_input_raw was rewound 110 times
        diehard_opso|   0|   2097152|     100|0.10490598|  PASSED
# The file file_input_raw was rewound 139 times
        diehard_oqso|   0|   2097152|     100|0.08537482|  PASSED
# The file file_input_raw was rewound 152 times
         diehard_dna|   0|   2097152|     100|0.22225323|  PASSED
# The file file_input_raw was rewound 154 times
diehard_count_1s_str|   0|    256000|     100|0.74983773|  PASSED
# The file file_input_raw was rewound 180 times
diehard_count_1s_byt|   0|    256000|     100|0.34313743|  PASSED
# The file file_input_raw was rewound 180 times
 diehard_parking_lot|   0|     12000|     100|0.25249632|  PASSED
# The file file_input_raw was rewound 181 times
    diehard_2dsphere|   2|      8000|     100|0.82852847|  PASSED
# The file file_input_raw was rewound 181 times
    diehard_3dsphere|   3|      4000|     100|0.92155762|  PASSED
# The file file_input_raw was rewound 229 times
     diehard_squeeze|   0|    100000|     100|0.00229783|   WEAK
# The file file_input_raw was rewound 229 times
        diehard_sums|   0|       100|     100|0.12110208|  PASSED
# The file file_input_raw was rewound 231 times
        diehard_runs|   0|    100000|     100|0.17842871|  PASSED
        diehard_runs|   0|    100000|     100|0.75467128|  PASSED
# The file file_input_raw was rewound 258 times
       diehard_craps|   0|    200000|     100|0.00000000|  FAILED
       diehard_craps|   0|    200000|     100|0.03461180|  PASSED
^C
pvar@dh-lab:~$
```

Το dieharder ενσωματώνει ένα ευρύτατο σύνολο στατιστικών ελέγχων και θεωρείται ως το κατ' εξοχήν εργαλείο για τους ελέγχους τυχαιότητας. Για την πραγματοποίηση ορισμένων ελέγχων απαιτεί τεράστιους όγκους δεδομένων. Γι' αυτό, υπήρξαν περιπτώσεις που "ανακύκλωσε" τους αριθμούς που του δώσαμε (χρησιμοποίησε τους ίδιους ξανά και ξανά). Παρ' όλα αυτά, τα αποτελέσματα που μας έδωσε ήταν ικανοποιητικότατα!

υψίσυχνες συνιστώσες. Κατ' επέκταση, ο υψηλός ρυθμός δειγματοληψίας δεν θα μας προσφέρει τίποτα. Τέλος, για την "αναλυτικότητα" των δειγμάτων θα επιλέξουμε τα 16bits, που προσφέρουν ικανοποιητική ακρίβεια χωρίς να φλερτάρουν με την υπερβολή.

Εμείς εργαστήκαμε σ' ένα σύστημα με τη διανομή CrunchBang, η οποία βασίζεται στο Debian. Έτσι, για την εγκατάσταση του εργαλείου δώσαμε, απλά:

```
sudo apt-get install sox
```

Στη συνέχεια, εφόσον το ραδιόφωνο είχε ρυθμιστεί κατάλληλα και είχε συνδεθεί στην είσοδο της κάρτας ήχου, ξεκινήσαμε την καταγραφή ως εξής:

```
rec -D -r 16k -c1 -b16 —comment '' rnd.u16
```

Με την εκτέλεση του παραπάνω, το rec άρχισε να καταγράφει τα παράσιτα του ραδιοφώνου στο αρχείο rnd.u16. Η παράμετρος –D απενεργοποιεί την εξομάλυνση (dithering) που πραγματοποιούν εξ ορισμού τα προγράμματα του πακέτου SoX. Παρατηρείστε επίσης το κενό string που έπεται της παραμέτρου comment (τα δύο απλά εισαγωγικά): Με αυτόν τον τρόπο εξασφαλίζουμε ότι το rec δεν θα προσθέσει κανένα σχόλιο κειμένου στο header του αρχείου rnd.u16.

Σ' αυτό το στάδιο της εργασίας, πρέπει να φροντίσουμε ώστε η εγγραφή να πραγματοποιείται χωρίς το λεγόμενο ψαλίδισμα (clipping). Το ψαλίδισμα αποτελεί ένα είδος παραμόρφωσης, που προκύπτει όταν η ένταση του "εισερχόμενου" σήματος υπερβαίνει το δυναμικό εύρος της βαθμίδας εισόδου. Για καλή μας τύχη, το προγραμματάκι rec εμφανίζει διαρκώς τη στάθμη του σήματος, ενώ μας ενημερώνει και για το αν πραγματοποιείται clipping. Στα δικά μας πειράματα χρειάστηκε να σταματήσουμε δυο-τρεις φορές τη διαδικασία και να την ξεκινήσουμε εκ νέου, αφού προηγουμένως προσαρμόσαμε κατάλληλα την ένταση του ραδιοφώνου. Όταν βεβαιωθήκαμε ότι λειτουργούσαν όλα σωστά, αφήσαμε τον υπολογιστή να καταγράψει το θόρυβο για μερικές ώρες.

"Απόσταξη"

Όταν τερματίσουμε την εγγραφή με το rec, θα έχουμε στη διάθεσή μας ένα αρχείο με δείγματα του ραδιοφωνικού θορύβου. Μη νομίζετε όμως ότι η εργασία μας έχει ολοκληρωθεί. Αυτό το αρχείο δεν περιέχει το πολυπόθητο σύνολο των τυχαίων αριθμών. Τα δείγματα που καταγράψαμε είναι σχεδόν βέβαιο ότι παρουσιάζουν μια κάποια συσχέτιση μεταξύ τους. Αυτό που πρέπει να κάνουμε τώρα είναι να διαχωρίσουμε τα ψήγματα πληροφορίας που μεταβάλλονται τυχαία, από εκείνα που παρουσιάζουν περιοδικότητα. Αυτή η διαδικασία είναι περίπλοκη και αποτελεί τη δυσκολότερη εργασία που πρέπει να πραγματοποιήσει μια γεννήτρια τυχαίων αριθμών.

Δεν ξέρουμε αν το έχετε αντιληφθεί, αλλά το αρχείο με τα δείγματα του ραδιοφωνικού θορύβου αποτελεί μια δεξαμενή εντροπίας. Από αυτήν θα *εξάγουμε* τους τυχαίους αριθμούς. Με άλλα λόγια, θα εργαστούμε περίπου με τον τρόπο που εργάζεται και ο πυρήνας του Linux. Η δική μας προσέγγιση, βέβαια, θα είναι μάλλον απλούστερη. Συγκεκριμένα, θα ακολουθήσουμε την εξής λογική: Κατ' αρχάς, θα απορρίψουμε τα περισσότερο σημαντικά (most significant) bits από κάθε δείγμα. Αυτό θα το κάνουμε γιατί τα συγκεκριμένα bits μεταβάλλονται με σχετικά μικρότερη συχνότητα και περιγράφουν τους ηχητικούς τόνους χαμηλής συχνότητας και μεγάλης έντασης, που ενδέχεται να περιλαμβάνει ο θόρυβος από το ραδιόφωνο. Αυτοί οι ηχητικοί τόνοι, όταν ακούμε τα παράσιτα, δεν διακρίνονται εύκολα μέσα στο

```
pvar@dh-lab:~$ sudo apt-get install haveged
[sudo] password for pvar:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  haveged
0 upgraded, 1 newly installed, 0 to remove and 2 not upgraded.
Need to get 29.5 kB of archives.
After this operation, 105 kB of additional disk space will be used.
Get:1 http://http.debian.net/debian/ wheezy/main haveged amd64 1.4-4 [29.5 kB]
Fetched 29.5 kB in 0s (151 kB/s)
Selecting previously unselected package haveged.
(Reading database ... 96575 files and directories currently installed.)
Unpacking haveged (from .../haveged_1.4-4_amd64.deb) ...
Processing triggers for man-db ...
Setting up haveged (1.4-4) ...
pvar@dh-lab:~$
```

Μπορεί να μην το έχετε ακούσει, όμως το εργαλείο haveged παρέχεται στα repositories όλων των βασικών διανομών Linux.

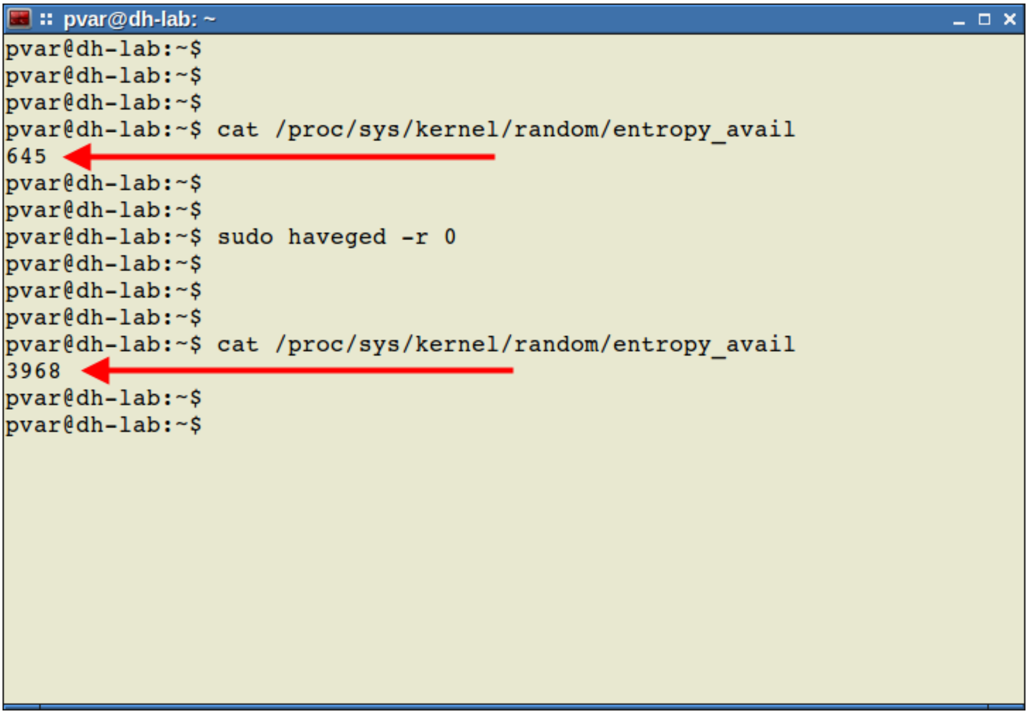
γενικότερο θόρυβο. Παρ' όλα αυτά, υπάρχουν και παρουσιάζουν περιοδικότητα. Για την εξαγωγή της "καθαρά" τυχαίας πληροφορίας, θα στραφούμε στα λιγότερα σημαντικά (less significant) bits. Θεωρητικά, θα μπορούσαμε να κρατήσουμε από κάθε δείγμα μόνο το bit 0 (αυτό με την μικρότερη αξία). Στην πράξη, όμως, θα χρειαστεί να το απορρίψουμε κι αυτό! Βλέπετε, το σήμα που παίρνουμε από το ραδιόφωνο είναι πολύ πιθανό να περιέχει μια συνιστώσα DC: Μια μικρή και σταθερή τάση, που έχει προστεθεί σε όλο το σήμα και κατ' επέκταση σε όλα τα δείγματα. Πρόκειται για ένα μέγεθος που περιγράφεται κι ως DC bias ή σκέτο bias. Αν σκέφτεστε ότι τελικά θα κρατήσουμε το δεύτερο bit, έχετε πέσει πολύ κοντά. Το δεύτερο bit επηρεάζεται ελάχιστα από το όποιο bias, αλλά δεν είναι απόλυτα ανεξάρτητο. Γι' αυτό το λόγο, θα εργαστούμε ως εξής: Θα συγκρίνουμε το δεύτερο bit ενός δείγματος, με το αντίστοιχο bit του αμέσως επόμενου δείγματος. Αν τα bits έχουν την ίδια τιμή, θα θεωρούμε ότι είναι επηρεασμένα από το bias, θα τα απορρίπτουμε και θα προχωράμε στο επόμενο ζεύγος δειγμάτων. Αν τα δύο bits είναι διαφορετικά, θα κρατάμε το ένα από τα δύο σαν καθαρό προϊόν τύχης και θα προχωράμε στο επόμενο ζεύγος δειγμάτων.

Αν θέλετε να εμβαθύνετε περισσότερο στην τεχνική που μόλις περιγράψαμε και στο όλο θέμα γενικότερα, μπορείτε να μελετήσετε το κείμενο "Randomness Recommendations for Security", από το Network Working Group του IETF (Internet Engineering Task Force). Το κείμενο βρίσκεται στη διεύθυνση ftp://ftp.ietf.org/rfc/rfc1750.txt, ενώ η συγκεκριμένη τεχνική αναπτύσσεται στην ενότητα "5.2.2 Using Transition Mappings to De-Skew".

Αυτοματοποιημένο ξεκαθάρισμα

Όταν τερματίσαμε την εγγραφή του ραδιοφωνικού θορύβου, είχε δημιουργηθεί ένα αρχείο δειγμάτων με μέγεθος κοντά στο 1GB. Από αυτόν τον τεράστιο όγκο δεδομένων θα κρατούσαμε ένα πολύ μικρό τμήμα, ακολουθώντας τη διαδικασία που περιγράψαμε παραπάνω. Έτσι, το επόμενο πράγμα που κάναμε ήταν να γράψουμε ένα μικρό πρόγραμμα σε Python. Ακολουθεί ο κώδικας (τον οποίο μπορείτε να κατεβάσετε κι από το `http://bit.ly/dh033distill`):

```
#!/usr/bin/python
with open('rnd.u16', 'rb') as input_file:
  # skip 64kB
  sample_bytes = input_file.read(65536)
  mask = 1
  rnd_data = 0
  byte_cnt = 0
  repeat = True
  output_file = open("rnd.bin", "ab")
  while repeat :
    two_sample_bytes = input_file.read(4)
    sample_bytes = bytearray(two_sample_bytes)
    if len(sample_bytes) == 4 :
      bit1 = sample_bytes[0] & 2
      bit2 = sample_bytes[2] & 2
      if bit1 != bit2 :
        if bit1 :
          rnd_data = rnd_data | mask
        mask = mask << 1
        if mask == 256 :
          mask = 1
          output_file.write(chr(rnd_data))
          byte_cnt = byte_cnt + 1
          rnd_data = 0
        if byte_cnt == 64000 :
          output_file.close()
          output_file = open("rnd.bin", "ab")
    else :
      repeat = False
```



Παρατηρείστε τα αποθέματα της δεξαμενής εντροπίας του συστήματος, πριν και μετά την εκκίνηση του haveged. Ο υπολογιστής μας δεν θα ξεμείνει ποτέ από τύχη.

Νομίζουμε ότι η λειτουργία του προγράμματος είναι αρκετά απλή. Αρχικά, ο κώδικας ανοίγει το αρχείο που γνωρίζουμε ότι περιέχει τα δείγματα θορύβου. Από αυτό διαβάζουμε τα πρώτα 64kB, τα οποία όμως δεν αξιοποιούμε πουθενά! Ουσιαστικά, αυτό που επιδιώκουμε εδώ είναι να "προσπεράσουμε" την περιοχή του αρχείου που φιλοξενεί το header, καθώς και τα πρώτα δείγματα θορύβου. Βλέπετε, όταν ξεκινήσαμε την εγγραφή και μέχρι να αφήσουμε το σύστημα στην ησυχία του, είχαμε κουνήσει το ραδιόφωνο και τα καλώδια αρκετές φορές. Το γεγονός αυτό ενδέχεται να εισήγαγε στο σήμα ορισμένες συνιστώσες με μικρό "χρόνο ζωής", αλλά μεγάλη ένταση. Αυτά τα παράσιτα που προστέθηκαν, αν και θα μπορούσαν να θεωρηθούν ως θόρυβος, ενδέχεται να παρουσιάζουν περιοδικότητα και να αλλοιώσουν την τυχαιότητα του θορύβου[2]. Λίγες γραμμές χαμηλότερα, ανοίγουμε και το αρχείο rnd.bin, στο οποίο θα τοποθετούνται οι τυχαίοι αριθμοί. Ακολουθεί ο κύριος βρόχος του προγράμματος, που εκτελείται μέχρι εξαντλήσεως των δειγμάτων θορύβου. Εντός του βρόχου διαβάζουμε κάθε φορά δύο διαδοχικά δείγματα (2 * 16 bits = 32 bits = 4 bytes). Στη συνέχεια απομονώνουμε το δεύτερο bit από το λιγότερο σημαντικό byte του κάθε δείγματος. Αν τα δύο bits είναι ίδια, τα ξεχνάμε και προχωράμε στο επόμενο ζεύγος δειγμάτων. Αν τα bits είναι διαφορετικά, κρατάμε το ένα από τα δύο και το "τοποθετούμε" σε ένα byte, εντός της μεταβλητής rnd_data. Κάθε φορά που αντιγράφουμε οκτώ τέτοια bits μέσα στο εν λόγω byte, το αποθηκεύουμε στο αρχείο rnd.bin και προχωράμε.

2. Εντάξει, πλέον έχετε κάθε λόγο να μας θεωρείτε παρανοϊκούς.

Έλεγχοι τυχαιότητας

Όταν ολοκληρώσαμε τη συγγραφή του προγράμματος, σπεύσαμε να το εκτελέσουμε. Εδώ οφείλουμε να σημειώσουμε ότι η γλώσσα C θα ήταν πολύ ταχύτερη κι άρα καταλληλότερη για την επεξεργασία των δειγμάτων. Ωστόσο, η Python αποτελεί την αγαπημένη μας επιλογή για τη συγγραφή προγραμμάτων "στα γρήγορα" και δεν καταφέραμε να της αντισταθούμε. Μετά από μερικά λεπτά[3] δημιουργήθηκε το rnd.bin. Μόλις είχαμε αποκτήσει περί τα 18MB τυχαίων αριθμών :D

Το πρώτο πράγμα που σκεφτήκαμε ήταν να ελέγξουμε "πόσο τυχαίοι" ήταν αυτοί οι αριθμοί. Έτσι, γράψαμε ένα ακόμα πρόγραμμα σε Python, που χρησιμοποιεί ένα μέρος των τυχαίων αριθμών για το σχηματισμό μιας εικόνας. Ο κώδικας βασίστηκε εξ ολοκλήρου σε συναρτήσεις της βιβλιοθήκης Image και δεν χρειάζεται καμία ιδιαίτερη επεξήγηση. Μπορείτε να τον κατεβάσετε από το http://bit.ly/dh033picit. Κρίνοντας από το γεγονός ότι η εικόνα που δημιουργήθηκε δεν περιελάμβανε κανένα επαναλαμβανόμενο μοτίβο, συμπεράναμε ότι οι αριθμοί μας ήταν αρκετά τυχαίοι. Φυσικά, γνωρίζαμε πολύ καλά ότι αυτός ο έλεγχος ήταν πρόχειρος κι εντελώς αναξιόπιστος. Ακριβώς γι' αυτό αποφασίσαμε να στραφούμε σε πιο σοβαρές λύσεις κι εγκαταστήσαμε τα προγράμματα ent, rng-tools και dieharder. Πρόκειται για εργαλεία που πραγματοποιούν διάφορους στατιστικούς ελέγχους και αποφαίνονται για το αν ένα σύνολο αριθμών παρουσιάζει τυχαία κατανομή. Για την εγκατάσταση των εργαλείων δώσαμε το εξής:

```
sudo apt-get install ent rng-tools dieharder
```

Αμέσως μετά, για την πραγματοποίηση των δοκιμών εκτελέσαμε διαδοχικά τα ακόλουθα:

```
ent rnd.bin
cat rnd.bin | rngtest
dieharder -a -g 201 -f rnd.bin
```

Εξετάζοντας τις σχετικές εικόνες του άρθρου, θα διαπιστώσετε ότι τα αποτελέσματα ήταν εντυπωσιακά. Οι αριθμοί που σχηματίσαμε από την επεξεργασία των δειγμάτων του ραδιοφωνικού θορύβου, ήταν "σχεδόν απόλυτα τυχαίοι"[4].

Μόνιμος μηχανισμός (όχι επιτήρησης)

Με τη διαδικασία που παρουσιάσαμε καταφέραμε να δημιουργήσουμε χειροκίνητα ένα τεράστιο σύνολο τυχαίων αριθμών. Με αυτούς θα μπορούσαμε να κατασκευάσουμε κλειδιά κρυπτογράφησης ή να "τροφοδοτήσουμε" κάποιο πρόγραμμα που απαιτεί μεγάλα ποσά εντροπίας. Δυστυχώς, όμως, δεν θα μπορούσαμε να τους αξιοποιήσουμε σε λειτουργίες που εκτελούνται αυτόματα κι επαναλαμβανόμενα, αφού το πλήθος τους δεν είναι ανεξάντλητο. Επομένως, αν για παράδειγμα επιθυμούμε να τροφοδοτήσουμε με εντροπία μια υπηρεσία του συστήματος, θα ήταν προτιμότερο να στραφούμε σε κάποιον άλλο μηχανισμό, όπως, π.χ., σε έναν που θα παράγει συνεχώς τυχαίους αριθμούς και, ιδανικά, θα τους διανέμει μέσω της δεξαμενής εντροπίας του συστήματος. Ε, λοιπόν, τέτοιες λύσεις υπάρχουν αρκετές. Κάποιες από αυτές στηρίζονται σε ειδικά προγράμματα και σε αντίστοιχες δικτυακές υπηρεσίες, οι οποίες παρέχουν τυχαίους αριθμούς στους πελάτες τους. Κάποιες άλλες στηρίζο-

3. Ναι, δεν πρόκειται για λάθος. Το προγραμματάκι μας αποδείχθηκε αρκετά αργόστροφο γι' αυτή την εργασία.
4. ...ό,τι κι αν μπορεί να σημαίνει αυτό ;)

νται σε αλγόριθμους παραγωγής ψευδοτυχαίων αριθμών, των οποίων τα προϊόντα θεωρούνται κατάλληλα για χρήση σε κρίσιμες εφαρμογές. Εμείς δοκιμάσαμε ένα εργαλείο που ανήκει στη δεύτερη ομάδα κι ακούει στο όνομα Haveged (`www.issihosts.com/haveged`).

Η εγκατάσταση στο σύστημά μας έγινε πανεύκολα:

```
sudo apt-get install haveged
```

Το συγκεκριμένο πρόγραμμα μπορεί να δημιουργήσει κατά παραγγελία ένα οποιοδήποτε "ποσό" τυχαίων αριθμών ή μπορεί να εκτελείται ως δαίμονας (daemon) και να τροφοδοτεί διαρκώς τη δεξαμενή εντροπίας του συστήματος. Εμείς, πριν εμπιστευτούμε τις ικανότητες του προγράμματος κι αρχίσουμε να τροφοδοτούμε τη δεξαμενή εντροπίας, αποφασίσαμε να πραγματοποιήσουμε μερικές δοκιμές. Έτσι, ξεκινήσαμε δημιουργώντας ένα αρχείο τυχαίων αριθμών μεγέθους 25M:

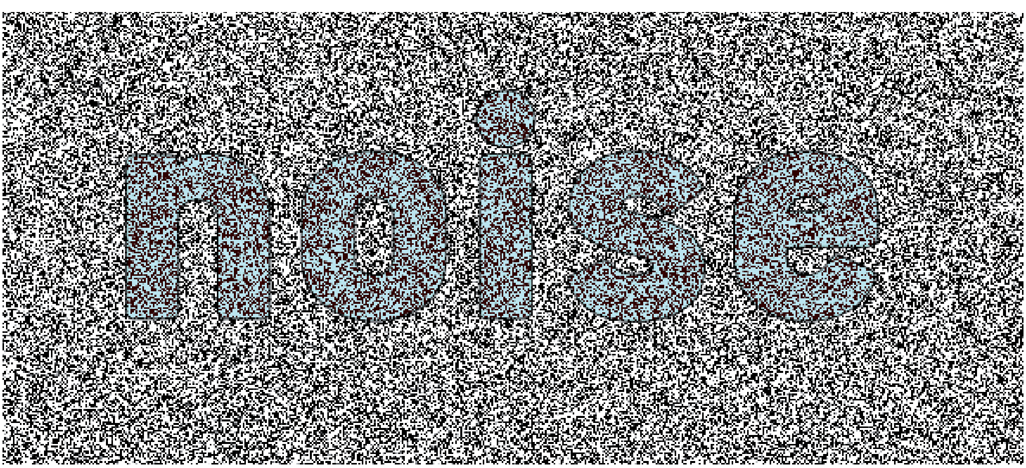
```
haveged -n 32M -f ./test.dat 2>/dev/null
```

Εκτελώντας το παραπάνω, οι τυχαίοι αριθμοί που παραγγείλαμε τοποθετήθηκαν στο αρχείο test.dat. Όπως και στην περίπτωση των δικών μας τυχαίων αριθμών, για τη διεξαγωγή των ελέγχων εκτελέσαμε τα ακόλουθα:

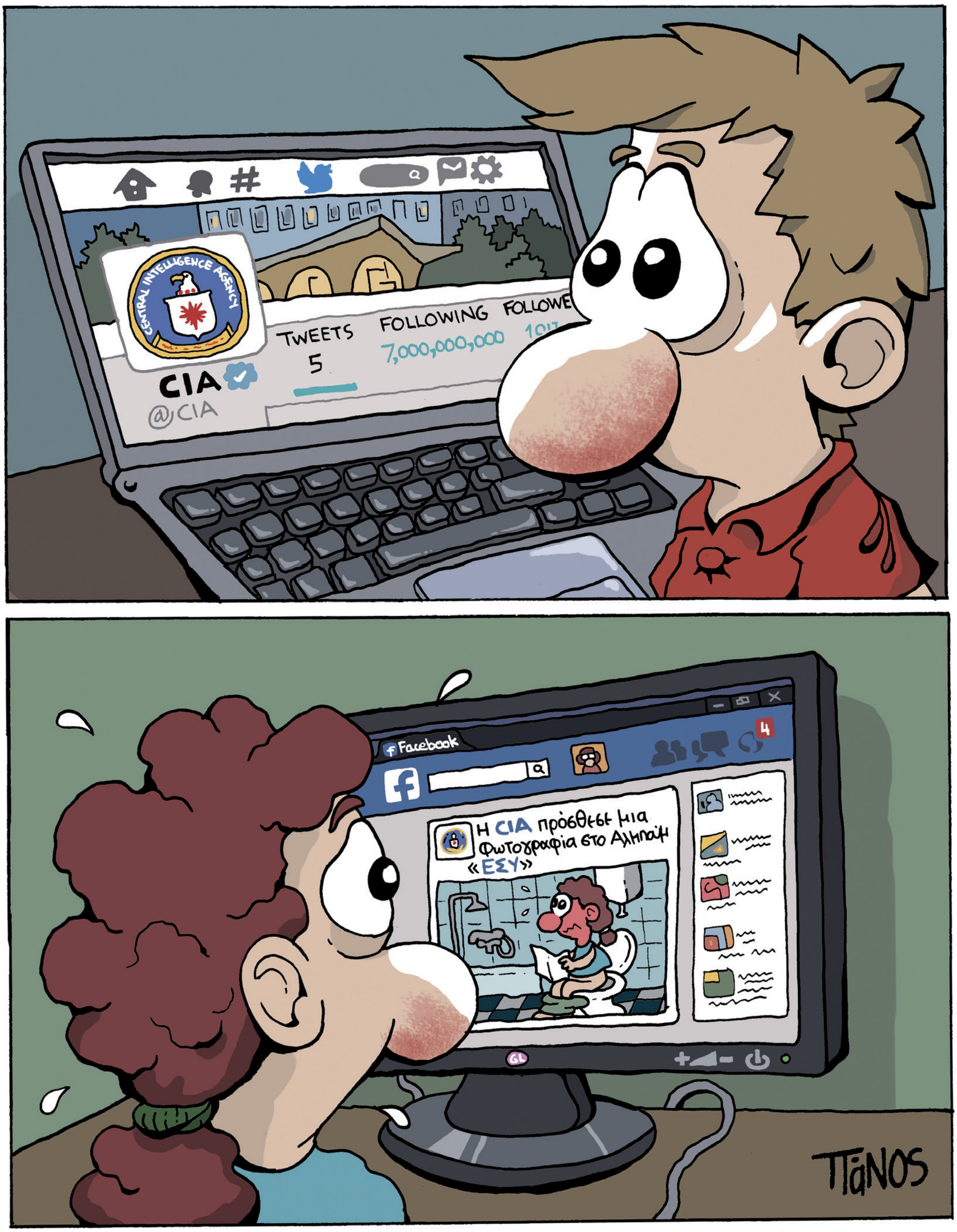
```
ent test.dat
cat test.dat | rngtest
dieharder -a -g 201 -f test.dat
```

Τα αποτελέσματα που λάβαμε ήταν ικανοποιητικότατα και μας ενέπνευσαν εμπιστοσύνη στις ικανότητες του haveged. Τέλος, ελέγξαμε το απόθεμα της δεξαμενής εντροπίας του συστήματος, ξεκινήσαμε το haveged ως δαίμονα (sudo haveged –r 0) και τσεκάραμε εκ νέου το μέγεθος της δεξαμενής. Περιττό να σας πούμε ότι, ξεκινώντας το haveged, η δεξαμενή γέμισε ακαριαία. Μετά κι από αυτό, ήμασταν σίγουροι ότι το σύστημά μας δεν θα ξέμενε ποτέ από εντροπία.

Εσείς τι θα κάνετε, τελικά; Θα αφήσετε το ζήτημα της τύχης στην τύχη; (Σ.τ.Ε. Είδατε τι έκανε εκεί; :P)



COOKIES


CENTRAL INTELLIGENCE AGENCY
TWEETS
5
FOLLOWING
7,000,000,000
CIA
@CIA
Facebook
Η CIA πρόσθεσε μια φωτογραφία στο Άλμπουμ «ΕΣΥ»